UNIVERSIDAD AUTÓNOMA DE MADRID

ESCUELA POLITÉCNICA SUPERIOR

Grado en Ingeniería Informática

DESARROLLO DE HERRAMIENTA PARA LA ANOTACIÓN MANUAL DE SECUENCIAS DE VÍDEO

-TRABAJO FIN DE GRADO-

Yoel Witmaar García
Tutor: Juan Carlos San Miguel Avedillo
Ponente: José María Martínez Sánchez
Junio 2015

DESARROLLO DE HERRAMIENTA PARA LA ANOTACIÓN MANUAL DE SECUENCIAS DE VÍDEO

Autor: Yoel Witmaar García
Supervisor: Juan Carlos San Miguel Avedillo
email: yoel.witmaar@estudiante.uam.es, juancarlos.sanmiguel@uam.es

Video Processing and Understanding Lab
Escuela Politécnica Superior
Universidad Autónoma de Madrid
Junio 2015

Resumen

Anotar imágenes y vídeos es algo importante para los que se dedican al tratamiento de imágenes, pero es una tarea muy dura para hacerla solamente de manera manual. Por eso es necesario desarrollar herramientas de anotación que faciliten este trabajo, de manera que el programa utilice algoritmos ya implementados que ayuden a hacer de la anotación de imágenes y videos una tarea más eficiente.

El primer paso de este trabajo es el de investigar las herramientas que hay en la actualidad y valorar las características que tienen. Utilizando esta información se pueden obtener ideas sobre cuáles son las características que más van a ayudar a los usuarios a anotar con más eficacia y a anotar cualquiera de los formatos de vídeo o imagen que haga falta.

El siguiente paso es elegir una herramienta de anotación de la que partir, una que por lo menos realice anotaciones manuales, para poder añadirle funcionalidades, como las anotaciones semiautomáticas y automáticas. También está como objetivo hacer que sea capaz de soportar más formatos de imagen o vídeo.

Lo más eficaz para aumentar la eficacia a la hora de anotar de una herramienta es añadirle anotaciones automáticas, de manera que se utilicen algoritmos de detección de objetos que ya están implementados. También es eficaz añadir mecanismos de propagación de anotaciones de una imagen a otra, los cuales pueden usar también algoritmos ya implementados.

Para este trabajo se ha escogido una herramienta de anotación de imágenes desarrollada por el Grupo de Tratamiento de Imágenes de la Escuela Politécnica Superior de la Universidad Autónoma de Madrid que es capaz de realizar anotaciones manuales, dibujando polígonos en la imagen y etiquetándolos, y de hacer presegmentación automática de la imagen, lo que quiere decir que puede utilizar un algoritmo que divida la imagen en regiones.

A partir de esto, lo que se han desarrollado son mejoras que permiten a la herramienta leer vídeos de fotograma en fotograma para anotarlos, en cualquier tipo de formato; hacer anotaciones automáticas con detectores de objetos, de manera que éstos detectan posibles objetos y el usuario elige los que le parece correctos para anotar, y utilizar un mecanismo manual para propagación de anotaciones de fotograma a fotograma dentro de un mismo vídeo, el cual consiste en mover una anotación mientras se van cambiando los fotogramas.

Palabras clave

Herramienta de anotación, anotación de vídeo, anotación de imagen, anotación manual, anotación semiautomática, anotación automática, propagación de anotaciones, región de una imagen, detector de objetos.

Abstract

Annotating images and videos is something important for those who work in image processing, but it is a hard task to do it only manually. That's why it's necessary to develop annotation tools which make this work easier, so that the program uses implemented algorithms which help to do the image and video annotation more efficient.

The first step of this work is to investigate actual tools and value their features. With this information you can get good ideas about which features are going to help the users more for annotating with more effectiveness or annotating any image or video format it's needed to.

The next step is choosing a tool for a start, one which performs manual annotations at least, so more functionalities can be added, like semiautomatic and automatic annotations. It is also an aim making the tool support more image and video formats.

The most effective thing for improving a tool's annotation efficiency is to add it automatic annotations which use implemented object detection algorithms. It's algo worth to add it annotation propagation mechanisms, which can use implemented algorithms too.

For this work an annotation tool developed by the Grupo de Tratamiento de Imágenes of the Escuela Politécnica Superior of the Universidad Autónoma de Madrid which is capable of perform manual annotations, drawing polygons on the image and labeling them, and do automatic presegmentation, which means that it can use an algorithm which divide the image in regions.

From this, improvements have been developed which allow the tool to read videos frame by frame for annotating them, on any format; to do automatic annotations using object detectors, so that they detect likely objects and the user can choose which are correct for annotating them, and using a frame by frame manual annotation propagation mechanism, which consists in moving an annotation while the frames are changing.

Keywords

Annotation tool, video annotation, image annotation, manual annotation, semiautomatic annotation, automatic annotation, annotation propagation, image region, object detector.

Índice general

Índice de ilustraciones

Índice de figuras

Índice de tablas

Glosario

GPL: General Public License

LGPL: Lesser General Public License

RDF: Resource Description Framework

RDFS: RDF Schema

JMF: Java Media Framework

ECL: Educational Community License

AFL: Academic Free License

DAML: DARPA Agent Markup Language

Checkbox: casilla de verificación

Edit mode: modo editar

1. Introducción

1.1. Motivación

Las anotaciones de imágenes o vídeos son importantes, ya que es la forma que existe para dejar constancia de lo que ocurre en ellos. Es una forma de organizarse necesaria para que sus usuarios puedan comprender mejor lo que ocurre en esas imágenes o vídeos.
En los últimos años se han desarrollado muchas herramientas de anotación semántica de imagen y/o vídeo, que han ido buscando cada vez mejorar calidad de las anotaciones y reducir el esfuerzo humano necesario para generar estas anotaciones.
Aumentar la eficiencia del usuario a la hora de anotar en imágenes o vídeos siempre ha sido siempre el objetivo prioritario, ya que realizar anotaciones manuales es una tarea larga y tediosa, especialmente cuando se trata de hacer anotaciones en vídeos con muchos fotogramas, en los que hay que anotar en cada uno de ellos.
La forma de aumentar esta eficiencia es mejorar las herramientas de anotación manual de imagen y/o vídeo para que también permitan realizar propagación de anotaciones y anotaciones automáticas:

- Hay varios mecanismos de propagación de anotaciones, como por ejemplo permitir realizar anotaciones semiautomáticas, que son aquellas que se generan a partir de una anotación manual realizada por el usuario en un fotograma utilizando algoritmos como la interpolación lineal o template matching [1]. También es posible utilizar un mecanismo manual en el cual el usuario arrastra la anotación de un objeto de un fotograma a otro y lo va ajustando a medida que este objeto cambia de posición y/o de tamaño.
- Las anotaciones automáticas son las que se generan a través de algoritmos de detección de objetos.

1.2. Objetivos

El primer objetivo de este trabajo es realizar una lista de herramientas de anotación semántica de imagen y/o vídeo desarrolladas hasta ahora, pudiendo ser de anotación manual o semiautomática, con sus características más interesantes. Tras esto, hay que decidir qué herramienta utilizar como base. Partiendo de esta herramienta, el resto de objetivos son:

- Conseguir que la herramienta sea capaz de leer y anotar en cualquier formato de imagen o vídeo.

- Mejorar la herramienta para que permita realizar anotaciones automáticas, utilizando algoritmos de detección de objetos ya implementados.
- Implementar un mecanismo de propagación de anotaciones en caso de que no lo hubiera, o modificarlo para que sea mejor en caso de que sí.

1.3. Estructura del documento

Capítulo 1: Introducción, motivación y objetivos

Capítulo 2: Estado del arte

Capítulo 3: Sistema inicial de anotación

Capítulo 4: Mejoras introducidas

Capítulo 5: Pruebas y validación

Capítulo 6: Conclusiones y trabajos futuros

Referencias

2. Estado del arte

Para mostrar el estado del arte se han hecho dos listas de herramientas: una de herramientas de anotación manual y otra de herramientas de anotación semiautomática. En ambas listas se muestra, para cada herramienta:

- Si se trata de una herramienta de anotación de imagen o de anotación de vídeo.
- Qué permite. Se refiere a la clase de anotaciones que permite según estos criterios [2]:
  - Tipo de metadatos: Se refiere al grado de la anotación. Se identifican los siguientes tipos:
    - Descriptivas: Contienen información proporcionada por el usuario sobre lo que hay en la imagen o vídeo.
    - Estructurales: Describen aspectos espaciales, temporales y espaciotemporales.
    - Multimedia: Se refieren a características de bajo nivel.
    - Administrativas: Contienen datos como la fecha de creación de la anotación, el nombre del creador, etc.
  - Granularidad de las anotaciones: Específica si las anotaciones describen todo el contenido o partes específicas de ella.
    - Si el contenido es una imagen, las anotaciones pueden referirse a la imagen entera o a regiones de ellas.
    - Si el contenido es un vídeo, las anotaciones pueden referirse al vídeo entero, a segmentos temporales, a fotogramas, a regiones dentro de un fotograma o incluso a regiones que se mueven a lo largo de varios fotogramas.
  - Localización manual de regiones: Se aplica solamente a herramientas que pueden anotar regiones, y se refiere a la forma en la que se pueden delimitar estas regiones.
  - Expresividad de las anotaciones: Es el nivel de expresividad que soporta el vocabulario de anotación. Hay herramientas que sólo permiten realizar anotaciones basadas en conceptos, y otras que además permiten realizarlas basadas en relaciones entre estos conceptos.
- La disponibilidad del código. En el caso de ser código abierto se muestra también la licencia que tiene.
- El lenguaje de programación en el que está escrito el código en caso de estar disponible.
- Los formatos soportados para la entrada de datos, es decir, los formatos de imagen o vídeo soportados.
- Los estándares de anotación que se emplean [2]:
  - Para la entrada: Se refiere a cómo se realizan las anotaciones, pudiéndose utilizar ciertos lenguajes de ontología soportados o texto libre y palabras clave.

- Para la salida: En qué formato se expresan las anotaciones generadas.

- Además, si la herramienta es de anotación semiautomática se muestra que característica o características son las que la convierten en tal.

2.1. Herramientas de anotación manuales

2.1.1. Listado

- KAT:
  - Es una herramienta de anotación de imagen. Su interfaz de usuario se muestra en la ilustración 1.
  - Permite anotaciones descriptivas y estructurales, y anotaciones basadas en conceptos, que pueden referirse a la imagen entera o a regiones específicas de ella. Las regiones se dibujan manualmente, pudiendo hacer rectángulos o polígonos [2].
  - El código está disponible bajo licencia LGPL.
  - El lenguaje de programación empleado es Java.
  - Los formatos de entrada de datos soportados son todos los formatos de imagen comunes, como por ejemplo JPG.
  - Los estándares de anotación empleados para la entrada son OWL y RDFS, y para la salida OWL [2].

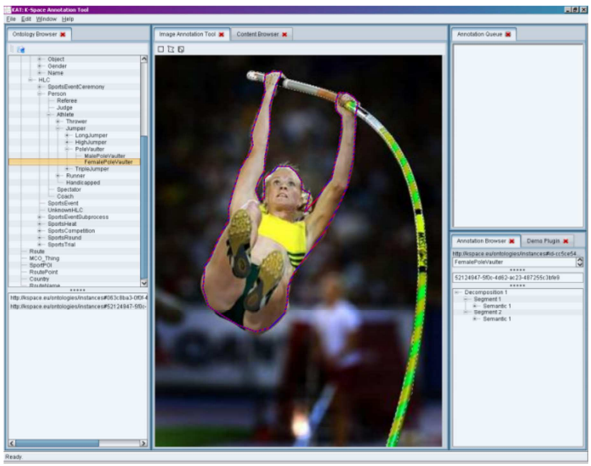

Ilustración 1: Ejemplo de anotación de imagen con KAT

- PhotoStuff:
  - Se trata de una herramienta de anotación de imagen, cuya interfaz es mostrada en la ilustración 2.
  - Son permitidas las anotaciones descriptivas, estructurales y administrativas, y las anotaciones basadas en conceptos y relaciones entre conceptos, que pueden referirse a la imagen entera o a regiones específicas de ella. Las regiones se delimitan con las herramientas de dibujo que realizan círculos, rectángulos y polígonos [2].

- El código es abierto, bajo la licencia Mozilla Public License Version 1.1 [3].
- El lenguaje de programación utilizado es Java [3].
- Los formatos de imagen soportados son aquellos que permiten incluir metadatos de una imagen dentro del fichero de la propia imagen, como por ejemplo JPEG y EXIF (cámaras).
- Los estándares de anotación utilizados para la entrada son OWL, RDF y RDFS, y para la salida RDF [2].

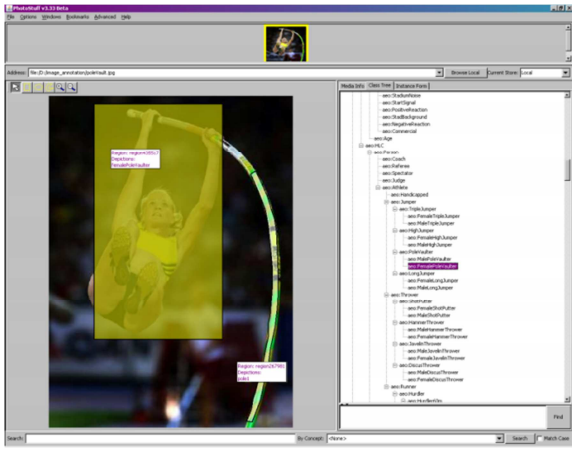

Ilustración 2: Ejemplo de anotación de imagen con PhotoStuff

- Caliph:
  - Esta herramienta sirve para anotar imágenes. Su interfaz consiste de dos paneles mostrados en la ilustración 3.
  - Permite anotaciones descriptivas, estructurales, administrativas y multimedia, y anotaciones basadas en conceptos y relaciones entre ellos, pero que sólo pueden referirse a la imagen entera, sin delimitar regiones específicas [2].
  - Es de código abierto, bajo licencia GPL [3].
  - El lenguaje de programación usado es Java [3].
  - El formato de entrada de datos soportado es JPEG [3].
  - Los estándares de anotación usados para la entrada son el texto libre y las palabras clave (keywords), y para la salida MPEG-7 [2].

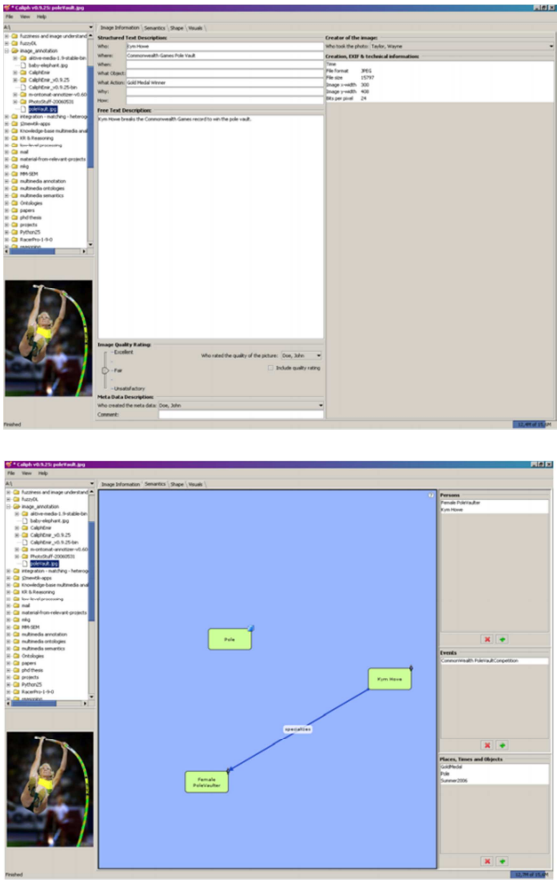

Ilustración 3: Ejemplo de anotación de imagen con Caliph: panel de información genérica de la imagen y panel de anotaciones semánticas

- SWAD:
  - Es una herramienta que anota imágenes. Un ejemplo de uso puede verse en la ilustración 4.
  - Son permitidas las anotaciones descriptivas y administrativas, y las anotaciones basadas en conceptos y sus relaciones, a nivel de imagen, sin especificar regiones de ella [2].
  - El código es abierto, bajo licencia W3Clicense.
  - El lenguaje de programación empleado es JavaScript [3].
  - El formato de entrada de imagen es JPEG [3].
  - Los estándares de anotación empleados para la entrada son el texto libre y las palabras clave (keywords), y para la salida RDF [2].

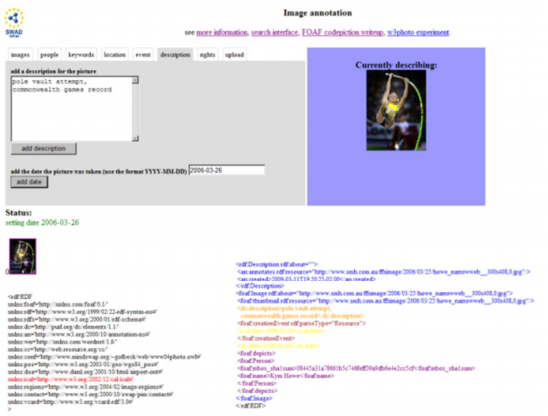

Ilustración 4: Ejemplo de anotación de imagen con SWAD

- LabelMe:
  - Esta herramienta es de anotación de imagen, y su interfaz nos la enseña la ilustración 5.
  - Permite anotaciones descriptivas, y anotaciones basadas en conceptos, las cuales pueden referirse a una región de la imagen dibujándola manualmente definiendo polígonos o a la imagen entera dibujando un polígono que englobe toda la imagen [2].
  - Es de código abierto y tiene una licencia GPL.
  - El lenguaje de programación usado es MATLAB.
  - El único formato de entrada de datos soportado es JPEG [3].
  - Los estándares de anotación usados para la entrada son el texto libre y las palabras clave (keywords), y para la salida XML personalizado [2].

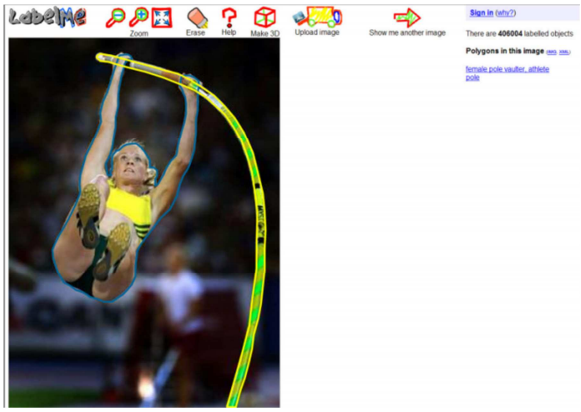

Ilustración 5: Ejemplo de anotación de imagen con LabelMe

- Ontolog:
  - Es una herramienta de anotación de vídeo. Un ejemplo de uso es mostrado en la ilustración 6.
  - Permite anotaciones descriptivas, estructurales y administrativas, y anotaciones basadas en conceptos y relaciones. Éstas pueden referirse a vídeos o a segmentos de vídeo. Permite extraer estadísticas simples,

como el porcentaje de tiempo que aparece un concepto en el vídeo [2].

- El código está disponible bajo petición.
- El lenguaje de programación utilizado es Java [3].
- Los formatos de entrada de datos soportados son aquellos que soportan los reproductores QuickTime y JMF [3].
- El estándar de anotación utilizado para la entrada es RDFS, y para la salida RDF [2].

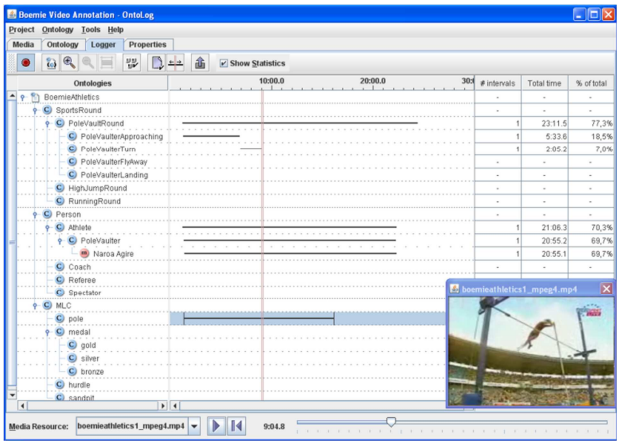

Ilustración 6: Ejemplo de anotación de vídeo con Ontolog

- Advene:
  - Se trata de una herramienta de anotación de vídeo. Para ver su interfaz de usuario hay que observar la ilustración 7.
  - Son permitidas las anotaciones descriptivas, estructurales y administrativas, y las anotaciones basadas en conceptos y relaciones, las cuales pueden referirse al video entero o a segmentos de él [2].
  - El código es abierto y la licencia es GPL [3].
  - El lenguaje de programación empleado es Python.
  - Soporta los formatos de video y audio soportados por el reproductor VLC player, incluyendo DVDs, como por ejemplo MPEG-1, MPEG-2, MPEG-4, DivX, mp3 y ogg [3].
  - El estándar de anotación empleado para la entrada es el texto libre, dentro de un formato específico, y para la salida XML personalizado [2].

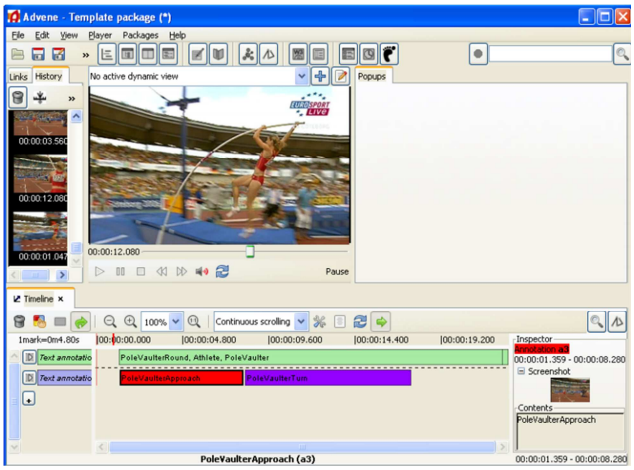

Ilustración 7: Ejemplo de anotación de vídeo con Advene

- Elan:
  - Esta herramienta es de anotación de vídeo. Puede verse un ejemplo de uso en la ilustración 8.
  - Permite anotaciones descriptivas, y anotaciones basadas en conceptos. Pueden referirse a videos enteros o a segmentos. Las anotaciones se pueden crear en diferentes niveles, que pueden estar interconectados, de manera que anotaciones de un nivel pueden referirse a las de otro nivel [2].
  - Es de código abierto y con una licencia GPL [3].
  - El lenguaje de programación empleado es Java [3].
  - Los formatos de entrada de datos soportados dependen del reproductor de video instalado. Los reproductores que soporta son Windows Media Player, QuickTime y JMF.
  - Los estándares de anotación empleados para la entrada son el texto libre y las palabras clave (keywords), y para la salida XML personalizado [2].

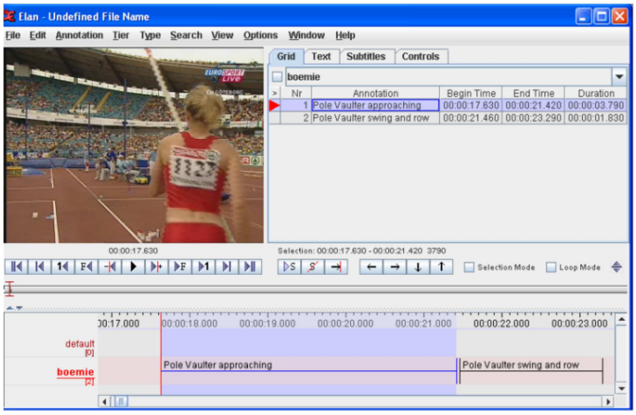

Ilustración 8: Ejemplo de anotación de vídeo con Elan

- Anvil:
  - Es una herramienta que anota vídeos y cuya interfaz gráfica se muestra en la ilustración 9.
  - Permite anotaciones descriptivas, estructurales y administrativas, y anotaciones basadas en conceptos. Pueden referirse a videos, segmentos de éstos o puntos específicos dentro de un vídeo. Permite niveles jerárquicos definidos por el usuario para las anotaciones, de la misma manera que Elan [2].
  - El código está disponible bajo petición, solamente para propósitos académicos y de investigación [3].
  - Está escrito en Java [3].
  - Soporta los formatos de entrada de datos AVI y MOV.
  - El estándar de anotación que se utiliza para la entrada es un esquema de XML definido por el usuario, y para la salida XML personalizado [2].

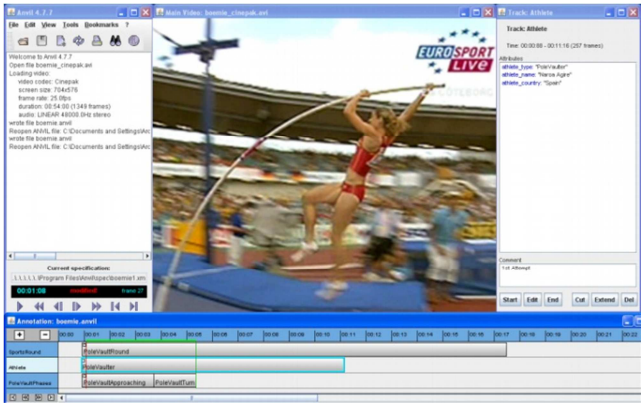

Ilustración 9: Ejemplo de anotación de vídeo con Anvil

2.1.2. Resumen

De las herramientas de anotación manuales, KAT, PhotoStuff, Caliph, SWAD y LabelMe son de imagen. Se puede decir que Caliph y SWAD son muy parecidas, ya que ambas permiten anotaciones sólo a nivel de imagen, mientras que el resto permite anotar regiones, pero lo compensan pudiendo realizar anotaciones basadas tanto en conceptos como en sus relaciones. De las herramientas que permiten anotar regiones KAT y LabelMe solamente realizan anotaciones basadas en conceptos, mientras que PhotoStuff es la herramienta de anotación de imagen más completa al permitir también anotaciones basadas en relaciones de conceptos y más tipos de anotaciones basadas en el tipo de metadatos. En cuanto a los formatos de imágenes soportados, KAT es la herramienta que más formatos soporta.
Por otro lado, Ontolog, Advene, Elan y Anvil son las herramientas de anotación de vídeo. Ninguna de ellas es capaz de anotar regiones, ni

quietas ni en movimiento. Ontolog y Advene son capaces de realizar anotaciones basadas en conceptos y relaciones, mientras que Elan y Anvil sólo basadas en conceptos. La herramienta que más formatos de vídeo soporta es Advene, al soportar los formatos que puede leer VLC player.
En la Tabla 1 se muestra una tabla que resume las características de estas herramientas.

Nombre de la herramienta	Contenido a anotar	Tipos de metadatos	Granularidad de las anotaciones	Localización manual de regiones	Expresividad de las anotaciones	Disponibilidad del código	Licencia	Lenguaje de programación	Formatos de entrada de datos	Estándares de anotación para la entrada	Estándares de anotación para la salida
KAT	imagen	descriptivo, estructural	imagen, región de la imagen	rectángulo, polígono	conceptos	disponible	LGPL	Java	formatos de imagen comunes	OWL/ RDFS	OWL
PhotoStuff	imagen	descriptivo, estructural, administrativo	imagen, región de la imagen	círculo, rectángulo, polígono	conceptos, relaciones	disponible	Mozilla Public License Version 1.1	Java	JPEG, EXIF	OWL/ RDF/ RDFS	RDF
Caliph	imagen	descriptivo, estructural, administrativo, multimedia	imagen	N/A	conceptos, relaciones	disponible	GPL	Java	JPEG	texto libre/ palabras clave	MPEG-7
SWAD	imagen	descriptivo, administrativo	imagen	N/A	conceptos, relaciones	disponible	W3C-license	JavaScript	JPEG	texto libre/ palabras clave	RDF
LabelMe	imagen	descriptivo	imagen, región de la imagen	polígono	conceptos	disponible	GPL	Matlab	JPEG	texto libre/ palabras clave	XML personalizado
Ontolog	vídeo	descriptivo, estructural, administrativo	vídeo, segmento de vídeo	N/A	conceptos, relaciones	disponible bajo petición	N/A	Java	formatos soportados por QuickTime y JMF	RDFS	RDF
Advene	vídeo	descriptivo, estructural, administrativo	vídeo, segmento de vídeo	N/A	conceptos, relaciones	disponible	GPL	Python	formatos soportados por VLC player	texto libre (dentro de un formato específico)	XML personalizado
Elan	vídeo	descriptivo	vídeo, segmento de vídeo	N/A	conceptos	disponible	GPL	Java	depende del reproductor instalado	texto libre/ palabras clave	XML personalizado
Anvil	vídeo	descriptivo, estructural, administrativo	vídeo, segmento de vídeo, puntos	N/A	conceptos	disponible bajo petición	N/A	Java	AVI, MOV	Esquema XML definido por el usuario	XML personalizado

Tabla 1: Sumario de características de herramientas de anotación manuales

2.2. Herramientas de anotación semiautomáticas

2.2.1. Listado

- AktiveMedia:
  - Es una herramienta de anotación de imagen. Para ver un ejemplo de uso se debe ir a la ilustración 10.
  - Permite anotaciones descriptivas, y anotaciones basadas en conceptos, que pueden referirse a la imagen o a regiones de ella. Las regiones se delimitan mediante rectángulos o círculos [2].
  - El código es abierto bajo licencias AFL, ECL y GPL [3].
  - El lenguaje de programación utilizado es Java [3].
  - Soporta todo tipo de formato de imagen: JPG, GIF, BMP, PNG, TIFF… [3].
  - Los estándares de anotación que se emplean para la entrada son los lenguajes ontológicos RDFS, OWL, DAML y DAML-ONT y el texto libre, y para la salida RDF [2].
  - Puede aprender durante el modo de anotación textual para poder hacer sugerencias al usuario después [2].

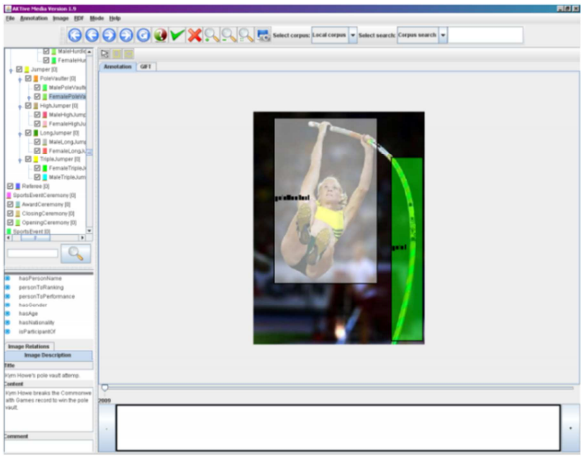

Ilustración 10: Ejemplo de anotación de imagen con AktiveMedia

- M-OntoMat-Annotizer:
  - La siguiente herramienta anota imágenes y tiene una interfaz como la mostrada en la ilustración 11.
  - Son permitidas las anotaciones descriptivas y multimedia, y las anotaciones basadas en conceptos, que pueden referirse a la imagen entera o a regiones específicas de ella. Las regiones se dibujan con herramientas manuales con rectángulos, elipses, polígonos y figuras pintadas a mano. Se pueden mezclar regiones [2].
  - El código está distribuido bajo licencia LGPL [3].
  - El código está escrito en Java [3].

- Soporta formatos de imagen soportados por JMF [3].
- Los estándares de anotación para la entrada son RDFS y DAML, y para la salida RDF [2].
- Para delimitar regiones específicas el usuario puede hacer uso de la segmentación automática que proporciona esta herramienta [2].

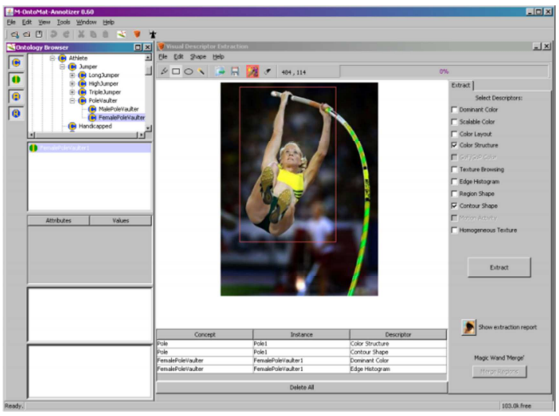

Ilustración 11: Ejemplo de anotación de imagen con M-OntoMat-Annotizer

- VIA:
  - Es una herramienta de anotación de vídeo. Su interfaz de usuario es como se muestra en la ilustración 12.
  - Permite anotaciones descriptivas, estructurales y administrativas, y anotaciones basadas en conceptos. Éstas pueden referirse a un fotograma o a regiones de éste, definidas dibujando figuras a mano, polígonos, círculos o rectángulos. También pueden referirse al vídeo entero, segmentos de éste o regiones que se mueven a lo largo de varios fotogramas [2].
  - Es de código abierto, bajo licencia GPLv2 [4].
  - El lenguaje de programación empleado es C++ [4].
  - Los formatos de entrada de datos soportados son MPEG-1 y MPEG-2.
  - Los estándares de anotación que se usan para la entrada son OWL y el texto libre, mientras que para la salida son XML y otros formatos de texto más comprensibles para los usuarios [2].
  - Se puede delimitar regiones de forma semiautomática, ya que la herramienta puede proveer la imagen segmentada ya y permitir al usuario corregirla mezclando regiones [2].

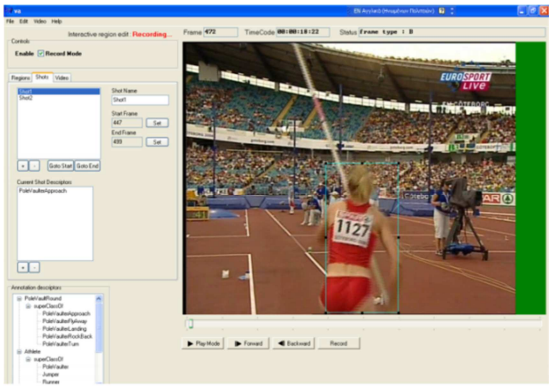

Ilustración 12: Ejemplo de anotación de vídeo con VIA

- VideoAnnEx:
  - Esta herramienta es de anotación de vídeo. Un ejemplo de anotación con esta herramienta puede verse en la ilustración 13.
  - Permite anotaciones descriptivas, estructurales y administrativas, y anotaciones basadas en conceptos y relaciones. Pueden referirse al video entero, a segmentos de éste, a un fotograma clave o a regiones específicas del fotograma, las cuales se dibujan usando rectángulos [2].
  - El código no está disponible.
  - Los formatos de entrada de datos soportados son MPEG-1 y MPEG-2.
  - Los estándares de anotación empleados para la entrada son XML y el texto libre, mientras que para la salida son MPEG-7 y XML [2].
  - Es capaz de aprender anotaciones, de manera que ayuda en la anotación buscando segmentos de vídeos similares al actual y etiquetándolo con las mismas descripciones [2].

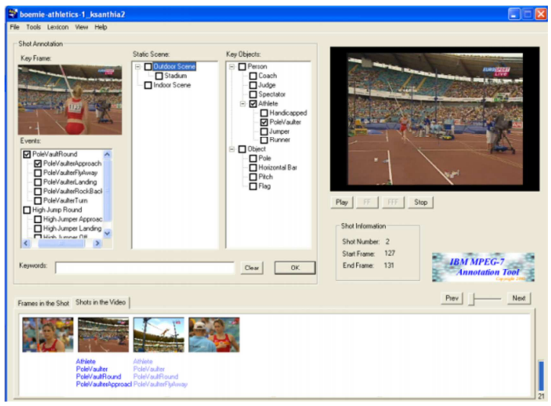

Ilustración 13: Ejemplo de anotación de vídeo con VideoAnnEx

- Semantic Video Annotation Suite (SVAS):
  - Se trata de una herramienta de anotación de vídeo. La interfaz gráfica de usuario de esta herramienta puede verse en la ilustración 14.

- Permite anotaciones descriptivas, estructurales y administrativas, y anotaciones basadas en conceptos. Pueden referirse a vídeos, segmentos de vídeo, fotogramas o regiones dentro de un fotograma. Las regiones se delimitan con rectángulos o polígonos [2].
- El código no está disponible.
- Los formatos de entrada de datos soportados son aquellos que soporta JMF.
- Los estándares de anotación utilizados para la entrada son el texto libre y las palabras clave (keywords), mientras que para la salida son MPEG-7 y XML [2].
- Se puede hacer segmentación automática de la imagen, además de que SVAS, una vez localizado un objeto de interés, es capaz de detectar objetos similares en todo el vídeo a través de un algoritmo de comparación [2].

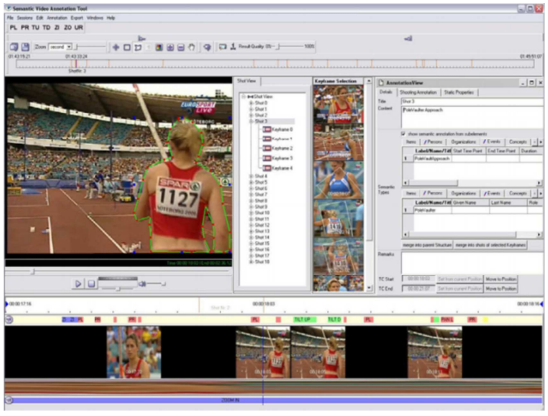

Ilustración 14: Ejemplo de anotación de vídeo con SVAS

- ViPER-GT:
  - Esta herramienta anota vídeos. Su interfaz es la que aparece en la ilustración 15.
  - Son permitidas las anotaciones descriptivas y estructurales, y las anotaciones basadas en conceptos. Pueden referirse a un fotograma o a una región, la cual es delimitada por rectángulos, elipses y polígonos. Las regiones pueden ser de un fotograma o de varios a la vez, moviéndose a lo largo de ellos [1, 2].
  - El código es abierto, con una licencia GPLv2 [4].
  - Está escrito en el lenguaje de programación Java [4].
  - Soporta los formatos de entrada de datos MPEG-1, MPEG-2 (en Windows) y algunos formatos adicionales si está instalado QuickTime para Java [3].
  - El estándar de anotación para la entrada utilizado en esta herramienta es el texto libre y las palabras clave, y para la salida el formato de archivo propietario ViPER, el cual está basado en XML [3].
  - Utiliza interpolación lineal para la propagación de anotaciones [1].

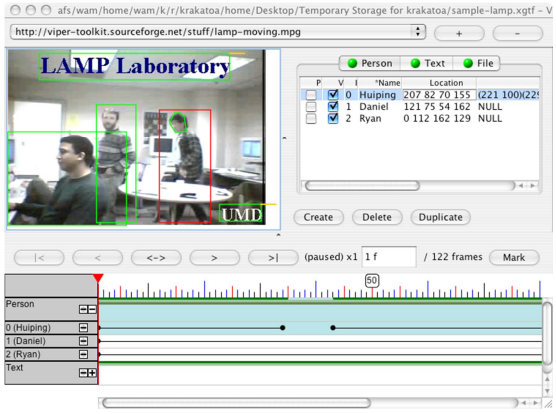

Ilustración 15: Ejemplo de anotación de vídeo con ViPER-GT

- iVAT:
  - La siguiente herramienta anota vídeos. Puede verse un ejemplo de uso de esta herramienta en la ilustración 16.
  - Es capaz de realizar anotaciones descriptivas y estructurales, y anotaciones basadas en conceptos y relaciones. Se pueden referir al vídeo, segmentos de vídeo, fotogramas, regiones dentro de un fotograma y a regiones en movimiento a lo largo de varios fotogramas. Las regiones se forman mediante rectángulos, elipses y polígonos [1].
  - El código no está disponible.
  - Soporta vídeos comprimidos en MPEG-4 [1].
  - Usa la lista de objetos como estándar de anotación para la entrada. Para la salida no se sabe [1].
  - Utiliza los algoritmos template matching e interpolación lineal para realizar las anotaciones semiautomáticas. Esta herramienta además permite las anotaciones automáticas, utilizando detectores de objetos con mecanismos de aprendizaje incremental [1].

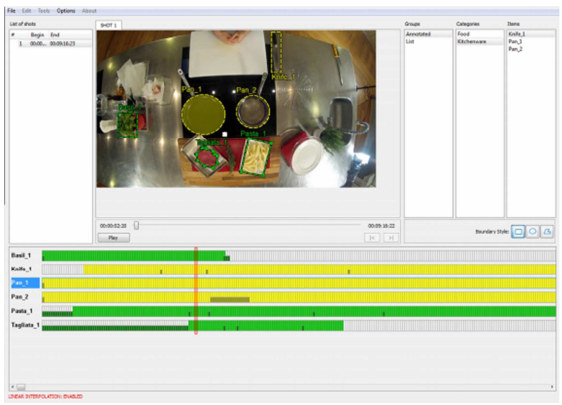

Ilustración 16: Ejemplo de anotación de vídeo con iVAT

2.2.2. Resumen

De las herramientas de anotación semiautomáticas, AktiveMedia y M-OntoMat-Annotizer son las de imagen. Ambas son muy parecidas, ya que permiten anotaciones basadas solamente en conceptos y que pueden referirse a la imagen o a regiones de ella. Ambas son capaces de soportar muchos formatos de imagen.
VIA, VideoAnnEx, SVAS, ViPER-GT e iVAT son las herramientas de anotación de vídeo. Todas ellas son capaces de anotar regiones de un fotograma, pero solamente VIA, ViPER-GT e IVAT permiten también las anotaciones de regiones en movimiento a lo largo de varios fotogramas. VideoAnnEx e iVAT son las únicas herramientas que permiten anotaciones basadas en relaciones. Y por último, en cuanto a formatos, SVAS y ViPER-GT son los que más soportan.
En la Tabla 2 se muestra una tabla que resume las características de estas herramientas.

Nombre de la herramienta	Contenido a anotar	Tipos de metadatos	Granularidad de las anotaciones	Localización manual de regiones	Expresividad de las anotaciones	Disponibilidad del código	Licencia	Lenguaje de programación	Formatos de entrada de datos	Estándares de anotación para la entrada	Estándares de anotación para la salida
AktiveMedia	imagen	descriptivo	imagen, región de la imagen	rectángulo, círculo	conceptos	disponible	AFL, ECL, GPL	Java	JPG, GIF, BMP, PNG, TIFF	RDFS/ OWL/ DAML/ DAML-ONT/ texto libre	RDF
M-OntoMat-Annotizer	imagen	descriptivo, multimedia	imagen, región de la imagen	rectángulo, elipse, polígono, figura pintada a mano	conceptos	disponible	LGPL	Java	formatos de imagen soportados por JMF	RDFS/ DAML	RDF
VIA	vídeo	descriptivo, estructural, administrativo	vídeo, segmento de vídeo, fotograma, región de un fotograma, región en movimiento a lo largo de varios fotogramas	figura pintada a mano, polígono, círculo, rectángulo	conceptos	disponible	GPLv2	C++	MPEG-1, MPEG-2	OWL/ texto libre	XML / formato de texto más comprensible para los usuarios
VideoAnnEx	vídeo	descriptivo, estructural, administrativo	vídeo, segmento de vídeo, fotograma, región del fotograma	rectángulo	conceptos, relaciones	no disponible	N/A	N/A	MPEG-1, MPEG-2	XML/ texto libre	MPEG-7/ XML
SVAS	vídeo	descriptivo, estructural, administrativo	vídeo, segmento de vídeo, fotograma, región del fotograma	rectángulo, polígono	conceptos	no disponible	N/A	N/A	formatos soportados por JMF	texto libre / palanras clave	MPEG-7/ XML
ViPER-GT	vídeo	descriptivo, estructural	fotograma, región del fotograma, región en movimiento a lo largo de varios fotogramas	rectángulos, elipses y polígonos	conceptos	disponible	GPLv2	Java	MPEG-1, MPEG-2, formatos adicionales si está instalado QuickTime para Java	texto libre / palanras clave	ViPER
iVAT	vídeo	descriptivo, estructural	vídeo, segmento de vídeo, fotograma, región de un fotograma, región en movimiento a lo largo de varios fotogramas	rectángulos, elipses y polígonos	conceptos, relaciones	no disponible	N/A	N/A	MPEG-4	lista de objetos	N/A

Tabla 2: Sumario de características de herramientas de anotación semiautomáticas

3. Sistema inicial de anotación

El sistema inicial de anotación elegido es una herramienta de anotación de imagen desarrollada por el Grupo de Tratamiento de Imágenes de la Escuela Politécnica Superior de la Universidad Autónoma de Madrid a partir de una herramienta de anotación semiautomática en Matlab desarrollada en la Universidad de California. Contiene una licencia que no permite una explotación comercial, pero sí para investigación [5].
Este programa necesita un archivo de texto que contenga las etiquetas, que definan a los posibles objetos, que se pueden asignar a las regiones, así que, una vez iniciado el programa, pulsando el botón label sel se podrá asignar a la región seleccionada únicamente una etiqueta elegida dentro de las que se encuentran en este fichero.
Delimitar y etiquetar regiones es la única forma de anotar en esta herramienta, por lo que todas las anotaciones son descriptivas y basadas en conceptos, y pueden referirse a la imagen o a regiones de ella. Las regiones pueden ser dibujadas manualmente mediante polígonos, pulsando el botón polygon, o creadas mediante una pre-segmentación automática de la imagen, pulsando el botón reset segm, pudiendo así mezclarlas para corregirlas, pulsando el botón merge sel.
Si después de cambiar la segmentación de la imagen, ya sea creando un nuevo polígono o haciendo una pre-segmentación, el usuario se arrepiente se puede deshacer solamente el último cambio realizado en la segmentación, pulsando el botón undo.
Los formatos de entrada de datos soportados son aquellos que soporta la función imread de Matlab: bmp, cur, gif, hdf, ico, jpeg, pbm, pcx, pgm, png, pnm, ppm, ras, tiff y xwd.
A la hora de guardar las regiones etiquetadas, lo cual se puede hacer pulsando el botón save, se generarán dos archivos para la descripción de la escena y otro auxiliar que sirva para recargar datos con la aplicación Matlab:

- Una imagen GIF que guarde la segmentación.
- Un archivo de texto de extensión .txt que contenga la descripción de las regiones de la imagen. Concretamente guardará el id de cada región junto con su etiqueta asignada.
- Un fichero auxiliar de extensión .labels.mat para guardar el valor de la variable que contiene las etiquetas utilizadas para las regiones y que será necesaria reutilizar con ese valor cuando se vuelva a cargar la herramienta Matlab.

La herramienta puede abrir una sola imagen o todas las imágenes que estén en un mismo directorio. En este último caso se abre la primera imagen al iniciar el programa y para pasar a la siguiente hay que pulsar el botón next, y luego para volver a la anterior el botón prev. Estos dos botones siempre preguntan al usuario si quiere guardar las anotaciones de la imagen actual antes de cambiar de imagen.

3.1. Inicialización del programa

Si se quiere iniciar el programa para anotar en una sola imagen hay que escribir en matlab el siguiente comando:
segmentation_ui(loaddir('./office_planta4', 'clean_scene_001.jpg'), loadstrings('SmartRoom_objects.txt'));
En el cual se llama a la función *segmentation_ui*, que tiene que recibir los siguientes argumentos:

- Un array de strings que guarde los nombres de los ficheros de las imágenes que se van a anotar. En el ejemplo se pasa como argumento una llamada a la función *loaddir*, la cual devuelve una lista de nombres de fichero que se encuentran en una determinada ruta y cumplen un patrón. Esta función recibe como parámetros:
  - Un string que contenga la ruta en la que se encuentran los ficheros. En el ejemplo se buscan ficheros que se encuentren en la ruta *'./office_planta4'*.
  - Un string que contenga el patrón que tienen que cumplir los ficheros. En el ejemplo se buscan ficheros que tengan el nombre *'clean_scene_001.jpg'*.
- Un array de strings que guarde todas las posibles etiquetas que se le pueden asignar a las regiones. En el ejemplo se pasa como argumento una llamada a la función *loadstrings*, la cual devuelve una lista de strings sacados de un archivo de texto. Esta función recibe como argumento el nombre del fichero de texto del que saca las posibles etiquetas.

En el caso de que se desee abrir todas las imágenes y moverse de una a otra sin volver a ejecutar se tendrá que escribir el siguiente comando:
segmentation_ui(loaddir('./office_planta4', '.jpg'), loadstrings('SmartRoom_objects.txt'));*
Una vez introducido el comando la aplicación abre la figura segmentation_ui.fig como interfaz de usuario.
Al abrirse la interfaz, se llama a la función *segmentation_ui_OpeningFcn*, que se encuentra en el fichero segmentation_ui.m, el cual guarda todas las funciones utilizadas por la interfaz. Esta función guarda los parámetros recibidos del programa en campos de la estructura de datos UI (estructura de datos de la interfaz de usuario), pone otros valores de la estructura por defecto y llama a la función *initialize_image*. El código de esta función puede verse en la figura 1.

```
function segmentation_ui_OpeningFcn(hObject, eventdata, handles, varargin)

handles.images     = varargin{1}; % first  parameter: images
handles.categories = varargin{2}; % second parameter: categories
handles.colormap = randomcolormap();
handles.oldsegm = 0;
handles.output = hObject; % Choose default command line output for segmentation_ui
guidata(hObject, handles); % Update handles structure

initialize_image(hObject,1);
```

Figura 1: Función segmentation_ui_OpeningFcn

La función *initialize_image* se encarga de inicializar la imagen que se va a mostrar en el hueco reservado para la imagen en la interfaz de usuario. Lo primero que hace es crear una presegmentación automática en cinco niveles que guarda en cinco ficheros de extensión .ppm. Después carga la segmentación de la imagen del fichero GIF, y, si no existe tal fichero, utiliza la presegmentación automática de nivel 1 como segmentación. Tras eso lee la imagen del fichero de imagen. Acto seguido lee las etiquetas guardadas en el fichero auxiliar de extensión .labels.mat en caso de existir el fichero. Por último, inicializa un par de valores más, como la matriz de regiones seleccionadas a ceros y la matriz de niveles de refinado a unos, y llama a la función *showimage*. Cabe destacar que todos estos valores cargados o inicializados han sido guardados en la estructura de datos UI. En la figura 2 aparece el código de esta función.

```
function initialize_image(handle,i)

d=guidata(handle);
d.i=i;

file = d.images{i};
%file = d.images(i,:);
precompute_segmentations(file);
d.segmentation= load_segmentation(file);
d.oldsegm     = 0;
d.oldrefine   = 0;
d.image       = imread(file);
if exist([file '.labels.mat'],'file')
  labels=load([file '.labels.mat']);
  d.labels = labels.labels;
else
  d.labels = struct([]);
end

l=cell(1);
for i=1:length(d.labels), l{i}=d.labels(i).label; end
set(d.listlabels,'String',l);
set(d.listlabels,'Value',1);

x=size(d.image,2);
y=size(d.image,1);
d.selstatus   = zeros(y,x,'uint8');
d.refinelevel = ones (y,x,'uint8');
d.red         = ones (y,x,'uint8');

guidata(handle,d);
showimage(d);
```

Figura 2: Función initialize_image

La función *showimage* es la que se encarga ya de mostrar la imagen. Primero visualiza la imagen junto con sus segmentos en caso de que esté marcado el checkbox de mostrar segmentaciones, el cual está por defecto lo está, y en caso de que no, solamente la imagen. Pone las regiones seleccionadas por el usuario en rojo. Al iniciar el programa no hay ninguna región seleccionada. Por último muestra en un texto encima de la imagen cuantos segmentos hay en la segmentación. La función *showimage* aparece en la figura 3.

```
function showimage(d)

if get(d.showsegmborders,'Value')==1
  temp=visualize_segm2(d.image,d.segmentation);
else
  temp=d.image;
end

sel=(d.selstatus==1);

temp(sel) = .3.*temp(sel)+.7.*256.*d.red(sel);
axes(d.img);
h=image(temp);
%h=imshow(temp);
set(h,'ButtonDownFcn',@img_ButtonDownFcn);
set(d.nsegm,'String',['now ',num2str( countsegments(d.segmentation) ),' segment(s)']);
```

Figura 3: Función showimage

3.2. *Los botones y sus funcionalidades*

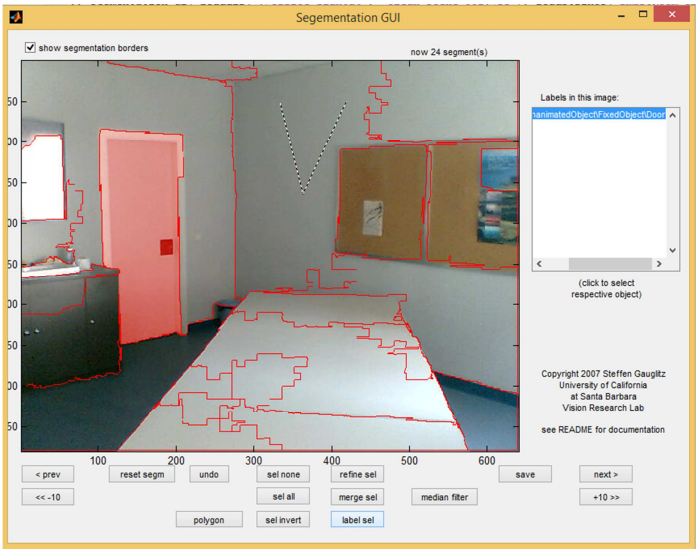

Ilustración 17: Interfaz de la herramienta de anotación inicial

Una vez iniciado el programa se muestra la interfaz de segmentación de imagen, la cual se muestra en la ilustración 17. En ella se encuentran, además de la imagen con su segmentación, los botones que se utilizan para llevar a cabo la tarea de la anotación. Los botones y sus funcionalidades son los siguientes:

3.2.1. Reset segm

Carga la presegmentación automática de nivel 1 del correspondiente fichero de extensión .ppm para utilizarla como segmentación de la imagen. Reinicia varios valores a valores por defecto: la matriz de regiones seleccionadas a ceros, la matriz de niveles de refinado a unos y la lista de etiquetas asignadas

a regiones vacía. Por último muestra los cambios realizados a través de una llamada a *showimage*. La función a la que se llama al pulsar este botón es la que aparece en la figura 4.

```
function reset_Callback(hObject, eventdata, handles)

d = guidata(gcbo);
d.oldsegm      = d.segmentation;
d.segmentation = load_presegmentation(d.images{d.i},1);
x=size(d.image,2);
y=size(d.image,1);
d.selstatus    = zeros(y,x,'uint8');
d.refinelevel  = ones (y,x,'uint8');
d.labels = struct([]);
guidata(gcbo,d);

showimage(d);
```

Figura 4: Función reset_Callback

3.2.2. Undo

Deshace el último cambio realizado en la segmentación, gracias a que cada vez que hay un cambio se guarda la segmentación anterior en la estructura de datos UI también. Tras eso llama a *showimage*, para mostrar el cambio. En la figura 5 se muestra el código ejecutado al pulsar este botón.

```
function undo_Callback(hObject, eventdata, handles)

d = guidata(gcbo);
if numel(d.oldsegm)>1
 d.segmentation=d.oldsegm;
 d.oldsegm=0;
end
if numel(d.oldrefine)>1
 d.refinement=d.oldrefine;
 d.oldrefine=0;
end
guidata(gcbo,d);

showimage(d);
```

Figura 5: Función undo_Callback

3.2.3. Next

Abre una ventana emergente que pregunta al usuario si quiere guardar las anotaciones de la imagen actual antes de pasar a la siguiente. Si el usuario pulsa ‘yes’ guarda utilizando la misma función que utiliza el botón save, y si pulsa ‘no’ no lo hace. En ambos casos procede a cambiar de imagen a la siguiente en la lista, mientras que si pulsa ‘cancel’ no lo hace. El cambio se realiza consultando la lista de ficheros de imagen abiertos para pasar al siguiente. También realiza una comprobación de que la imagen actual no es la última de la lista, y si lo es muestra un mensaje en una ventana advirtiéndolo al usuario y borra todos los ficheros de extensión .ppm que

guarden presegmentaciones automáticas. Si no lo es, llama a la función *initialize_image* para inicializar la nueva imagen. Su función de activación aparece en la figura 6.

```
function next_Callback(hObject, eventdata, handles)

answer=questdlg('save before going to next image?','save?');
if strcmp(answer,'Cancel'), return, end
if strcmp(answer,'Yes'), save_Callback(handles.save,[],handles); end

d = guidata(gcbo);

i = d.i + 1;
if( i > length(d.images) )
  msgbox('End of list reached! You may close the application.', ...
                'End of list','none','modal');

                %% delete all the segmentation !!rely on the fact the user did
                %% NOT name his/her files with *.segpre*.ppm & name them "name" . "ext"!!

   for i=1:length(d.images)
     file = d.images{i};
     delete([ file '.segpre*.ppm']);
   end

  return
end

initialize_image(gcbo,i);
```

Figura 6: Función next_Callback

3.2.4. Show segmentation borders

Se trata de un checkbox, que cuando está marcado muestra las segmentaciones en la imagen, y cuando no lo está no lo hace. Cuando se pulsa sobre él simplemente se llama a la función *showimage*, donde ya se hace esta comprobación. El código de la función ejecutada al pulsar este checkbox se muestra en la figura 7.

```
function showsegmborders_Callback(hObject, eventdata, handles)

d = guidata(gcbo);
showimage(d);
```

Figura 7: Función showsegmborders_Callback

3.2.5. La imagen

Tampoco es un botón, pero si se pulsa en ella se selecciona la región en la que se encuentra el punto donde se ha hecho clic en caso de no estar seleccionada, y se deselecciona en caso de que sí. Esta región es localizada gracias a la matriz de segmentación y las coordenadas del punto, y se cambia la matriz de regiones seleccionadas poniendo los valores que corresponden a esa región a uno en caso de ser cero y a cero en caso de ser uno. Tras llamar a *showimage* la región se muestra de color rojo en la imagen si ha sido seleccionada o deja de estar roja en caso de que haya sido deseleccionada.

La función que aparece en la figura 8 es la que se activa cuando se pulsa la imagen.

```
function img_ButtonDownFcn(hObject, eventdata, handles)

d=guidata(gcbo);
pt = get(d.img, 'CurrentPoint');
x = int32(pt(1,1)); y = int32(pt(1,2));

clickedtoken=d.segmentation(y,x);
oldstatus   =d.selstatus(y,x);
newstatus   =1-oldstatus;

d.selstatus(d.segmentation == clickedtoken) = newstatus;
guidata(hObject,d);

showimage(d);
```

Figura 8: Función img_ButtonDownFcn

3.2.6. Prev

Hace lo mismo que el botón next, sólo que en lugar de ir a la siguiente imagen accede a la anterior. En caso de que se encuentre ya en la primera imagen solamente muestra el mensaje de advertencia en una ventana y no borra los ficheros de presegmentación automática. El código de la función a la que llama este botón al pulsarse aparece en la figura 9.

```
function prev_Callback(hObject, eventdata, handles)

answer=questdlg('save before going to previous image?','save?');
if strcmp(answer,'Cancel'), return, end
if strcmp(answer,'Yes'), save_Callback(handles.save,[],handles); end

d = guidata(gcbo);

i = d.i - 1;
if( i < 1 )
  msgbox('Start of list reached!','Start of list','none','modal');
  return
end
```

Figura 9: Función prev_Callback

3.2.7. Save

Guarda todos los datos necesarios para las anotaciones en sus ficheros. Guarda la segmentación en el fichero GIF, la lista de etiquetas asignadas en el fichero auxiliar de extensión .labels.mat y la descripción de las regiones en el fichero de extensión .txt. Este último lo hace llamando a la función *buildSceneDescription*, que se encarga de imprimir en el fichero en un formato específico. La figura 10 muestra la función de activación de este botón.

```
function save_Callback(hObject, eventdata, d)

[map,s] = convert2_8bit(d.segmentation);
imwrite( s, [d.images{d.i} '.seg.gif'] );

j=1;
savelabels=0;
for l=1:length(d.labels) % only save non-empty tokens! (use for display, too?!)
  mylabel=d.labels(l).segmenttoken;
  if( numel(find(d.segmentation==mylabel,1)) )>0
    savelabels=1;
    labels(j)=d.labels(l);
    labels(j).segmenttoken=map( labels(j).segmenttoken+1 ); % and, very important: map to uint8 values!
    j=j+1;
  end
end

if savelabels ~= 1
  warning('segmentation_ui:save_Callback','No labels saved!');
else
  save([d.images{d.i} '.labels.mat'],'labels');
end

buildSceneDescription([d.images{d.i} '.sceneDescription.txt'], [d.images{d.i} '.labels.mat'], [d.images{d.i} '.seg.gif']);
```

Figura 10: Función save_Callback

3.2.8. Merge sel

Fusiona las regiones seleccionadas como una sola región. Lo hace poniéndoles el mismo token de segmentación, es decir, id de región, a todas las regiones. Para hacer el cambio visible se llama a *showimage*. En la figura 11 aparece la función que se ejecuta.

```
function merge_Callback(hObject, eventdata, d)

d.oldsegm=d.segmentation;
newtoken=d.segmentation( find(d.selstatus==1,1) );
d.segmentation( d.selstatus==1 ) = newtoken;
guidata(hObject,d);
showimage(d);
```

Figura 11: Función merge_Callback

3.2.9. Refine sel

Aumenta el nivel de refinado de una o varias regiones seleccionadas, es decir, las divide en regiones más pequeñas. Si no hay regiones seleccionadas o las regiones seleccionadas tienen un nivel 5 de refinado no hace nada. En caso de continuar, cambia la matriz de niveles de refinado para aumentar en uno el nivel de refinado de las regiones seleccionadas y carga la presegmentación automática del nuevo nivel para usarla como segmentación en estas regiones. Utiliza *showimage* para la actualización visual. Para ver la función llamada por este botón hay que observar la figura 12.

```
function refine_Callback(hObject, eventdata, d)

sel = d.selstatus==1;
if ~any(sel(:)), return, end
refinementlevel = min( d.refinelevel(sel))+1;
if refinementlevel>5, return, end % can't refine any further

d.oldrefine=d.refinelevel;
d.oldsegm  =d.segmentation;

d.refinelevel(sel)=refinementlevel;
segm=load_presegmentation(d.images{d.i},refinementlevel);
d.segmentation(sel)=segm(sel);

guidata(hObject,d);

showimage(d);
```

Figura 12: Función refine_Callback

3.2.10. Label sel

Asigna una etiqueta de las que se encuentran en el fichero de posibles etiquetas a una o varias regiones seleccionadas. Lo hace abriendo una ventana emergente en la que pregunta al usuario que etiqueta quiere utilizar dentro de las que ha leído del fichero de texto. Si el usuario pulsa ‘ok’ se comprueba si alguna de las regiones tenía una etiqueta asignada ya. En ese caso su id de región se encontrará en la lista de etiquetas asignadas y sólo habrá que cambiar su etiqueta, y en caso contrario habrá que añadir un nuevo elemento a la lista con el id de la región nueva y su etiqueta. El código de la función de activación de este botón es el mostrado en la figura 13.

```
function label_Callback(hObject, eventdata, d)

segments=deleteduplicates(d.segmentation(d.selstatus == 1));
if numel(segments)<1, return, end

[sel,ok]=listdlg('ListString',d.categories,'SelectionMode','single', ...
    'ListSize',[250 500],'Name','select label');
if ok ~= 1, return, end % user clicked 'cancel'

answer=getHierarchicalLabel(d.categories,sel);

%answer=inputdlg({'Enter label:'},'Label',1,{'<label>'});
%if numel(answer) ~= 1, return, end  % user clicked 'cancel'

for s=1:length(segments) % for all selected segments

  i=1;
  while i<=length(d.labels) % already an entry existing?
    if d.labels(i).segmenttoken==segments(s), break, end
    i=i+1;
  end % if loop runs through, i is automatically pointing to next entry

  d.labels(i).segmenttoken = segments(s);
  d.labels(i).label        = answer;
end

for i=1:length(d.labels), l{i}=d.labels(i).label; end
set(d.listlabels,'String',l);
guidata(hObject,d);
```

Figura 13: Función label_Callback

3.2.11. Sel none

Deselecciona todas las regiones de la imagen poniendo todos los valores de la matriz de regiones seleccionadas a cero. Después llama a *showimage*. En la figura 14 puede verse el código de la función que se ejecuta al pulsar el botón.

```
function selnone_Callback(hObject, eventdata, d)

d.selstatus(:)=0;
guidata(hObject,d);
showimage(d);
```

Figura 14: Función selnone_Callback

3.2.12. Sel all

Hace lo contrario al botón sel none. Selecciona todas las regiones de la imagen poniendo todos los valores de la matriz de regiones seleccionadas a uno y llamando a *showimage*. La función de activación es la que enseña la figura 15.

```
function selall_Callback(hObject, eventdata, d)

d.selstatus(:)=1;
guidata(hObject,d);
showimage(d);
```

Figura 15: Función selall_Callback

3.2.13. Sel invert

Selecciona las regiones deseleccionadas y deselecciona las seleccionadas. Simplemente lee los valores de la matriz de regiones seleccionadas y cambia los unos por ceros y los ceros por unos. Al final llama a *showimage*. Para ver el código de la función de activación de este botón hay que ir a la figura 16.

```
function selinvert_Callback(hObject, eventdata, d)

d.selstatus=1-d.selstatus;
guidata(hObject,d);
showimage(d);
```

Figura 16: Función selinvert_Callback

3.2.14. -10

Hace lo mismo que el botón prev, sólo que en lugar de retroceder una imagen retrocede diez. Este botón tiene su función de activación mostrada en la figura 17.

```
function prev10_Callback(hObject, eventdata, handles)

answer=questdlg('save before going to 10th previous image?','save?');
if strcmp(answer,'Cancel'), return, end
if strcmp(answer,'Yes'), save_Callback(handles.save,[],handles); end

d = guidata(gcbo);

i = d.i - 10;
if( i < 1 )
  msgbox('Start of list reached!','Start of list','none','modal');
  return
end

initialize_image(gcbo,i);
```

Figura 17: Función prev10_Callback

3.2.15. +10

Hace lo mismo que el botón next, sólo que en lugar de avanzar una imagen avanza diez. La figura 18 es la que enseña la función a la que llama este botón.

```
function next10_Callback(hObject, eventdata, handles)

answer=questdlg('save before going to 10th next image?','save?');
if strcmp(answer,'Cancel'), return, end
if strcmp(answer,'Yes'), save_Callback(handles.save,[],handles); end

d = guidata(gcbo);

i = d.i + 10;
if( i > length(d.images) )
  msgbox('End of list reached!','End of list','none','modal');
  return
end

initialize_image(gcbo,i);
```

Figura 18: Función next10_Callback

3.2.16. Median filter

Abre una ventana emergente que pregunta al usuario que tamaño de kernel quiere utilizar para la mediana. Después aplica el filtro a la segmentación de la imagen y utiliza *showimage* para mostrar el cambio. Al pulsarse este botón se llama a la función mostrada en la figura 19.

```
function median_Callback(hObject, eventdata, d)

answer=inputdlg({'Enter kernel size:'},'Kernel size',1,{'5'});
if numel(answer) ~= 1, return, end  % user clicked 'cancel'

k=str2double(answer{1});
d.oldsegm=d.segmentation;
d.segmentation = medfilt2(d.segmentation,[k k],'symmetric');
guidata(hObject,d);
showimage(d);
```

Figura 19: Función median_Callback

3.2.17. La lista de etiquetas

Si se pulsa una de las etiquetas de esta lista se selecciona la región que corresponde con la etiqueta pulsada. Esto lo hace mirando en la lista de etiquetas asignadas cual es la id de región de la etiqueta pulsada, para poder buscar esa región en la matriz de segmentación y cambiar la matriz de regiones seleccionadas. Al igual que casi todas las funciones, llama a *showimage* para que el cambio se haga visible. La función que aparece en la figura 20 es la función que se ejecuta al pulsar una de las etiquetas asignadas a regiones de la lista.

```
function listlabels_Callback(hObject, eventdata, d)

s=get(hObject,'Value');
if numel(s)~=1 || s>length(d.labels), return, end
select=d.labels(s).segmenttoken;
d.selstatus(:)=0;
d.selstatus(d.segmentation == select) = 1;
guidata(hObject,d);

showimage(d);
```

Figura 20: Función listlabels_Callback

3.2.18. Polygon

Permite al usuario delimitar una región en la imagen manualmente creando un polígono. Esto lo hace trazando líneas entre los puntos que va pulsando el usuario y una última línea entre el primer y último punto. Así se crea una región con la forma de este polígono y se le encuentra un token de segmentación o id de región que no se esté utilizando y que se pueda incluir en la segmentación de la imagen. Finalmente, usa la función *showimage*. El código de la figura 21 corresponde a la función de activación de este botón.

```
function polygon_Callback(hObject, eventdata, d)

[ly,lx,depth]=size(d.image);

[px,py]=getline(d.img);
X=ones(ly,1)*(1:lx);
Y=(1:ly).'*ones(1,lx);
in=inpolygon(X,Y,px,py);

d.oldsegm=d.segmentation;
d.segmentation(in)=findunusedtoken(d.segmentation);
guidata(hObject,d);

showimage(d);
```

Figura 21: Función polygon_Callback

En la figura 22 se muestra el diagrama de bloques de esta herramienta, la cual enseña todas sus funcionalidades conectadas.

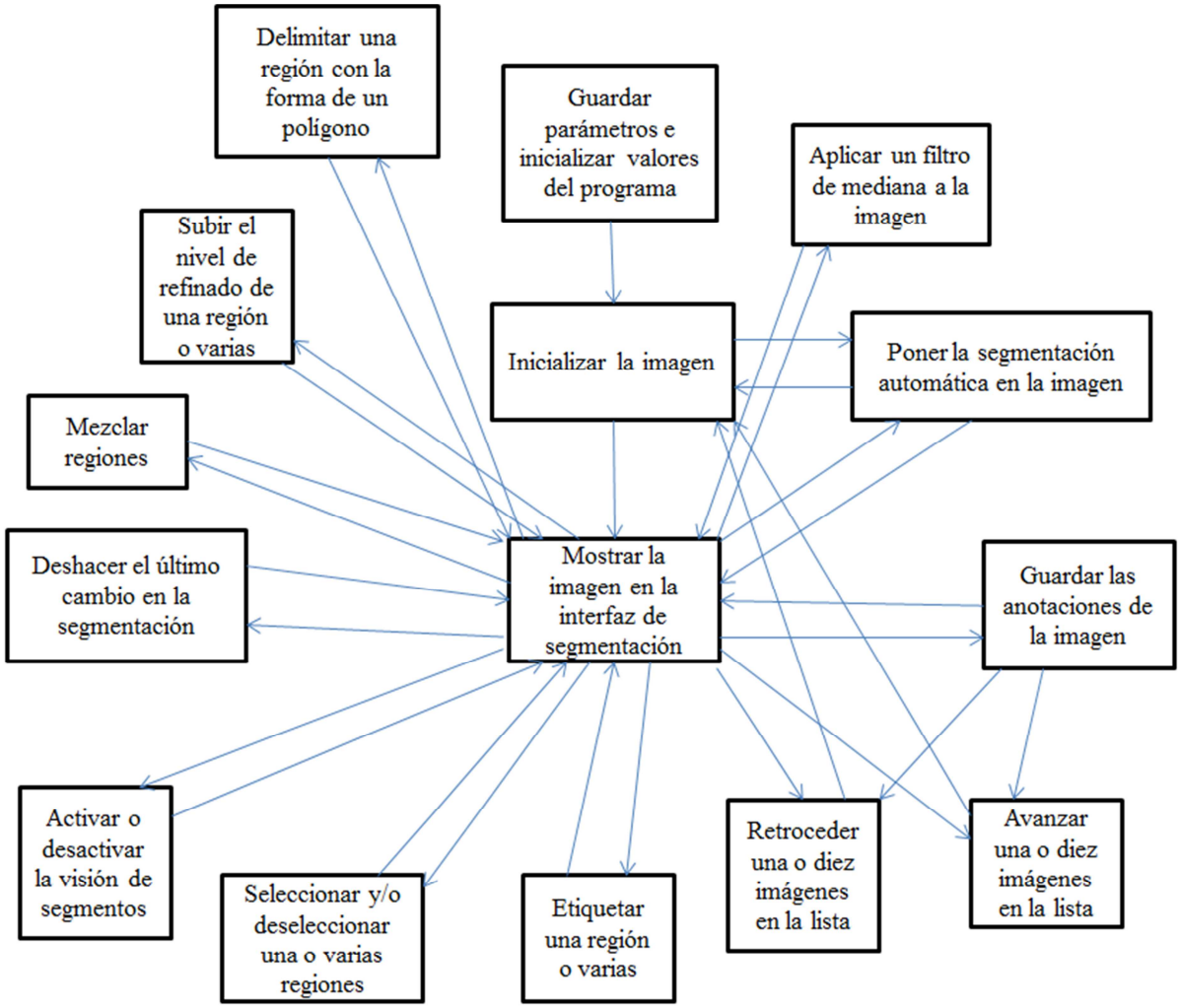

Figura 22: Diagrama de bloques del sistema de anotación inicial

4. Mejoras introducidas

Con las mejoras introducidas la interfaz de segmentación ha cambiado. Se han añadido nuevos botones en la derecha de la figura. Se puede ver la nueva interfaz en la ilustración 18.

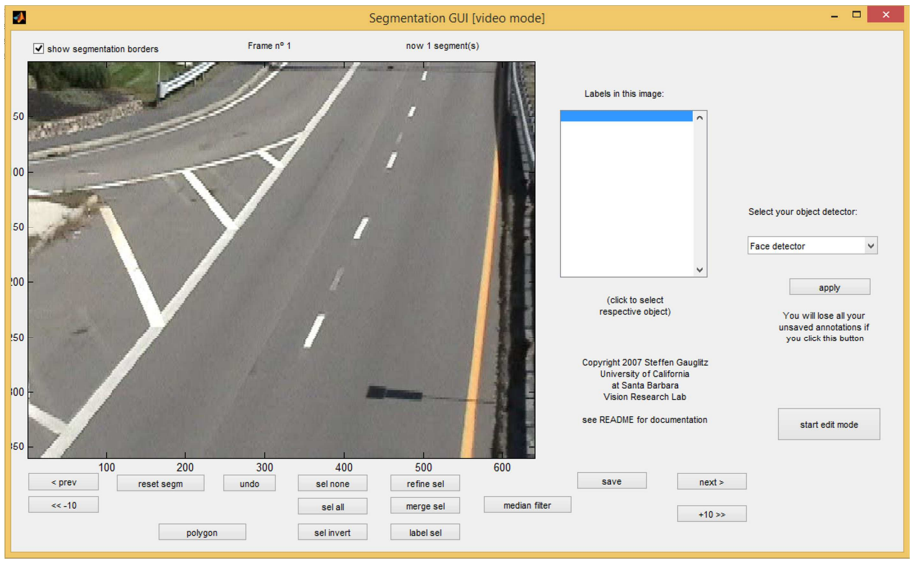

Ilustración 18: Interfaz de segmentación de la herramienta de anotación final

4.1. Lectura de varios formatos de vídeo

A la herramienta se le ha añadido un nuevo modo: el modo vídeo. En este modo se lee un vídeo y se muestran los fotogramas uno a uno, pudiendo navegar a través de ellos con los botones next y prev, como si se trataran de imágenes distintas. La diferencia está en que estos fotogramas forman parte de un mismo vídeo, por lo que los datos de todos ellos deben guardarse en los mismos ficheros. Para poder saber qué datos corresponden a cada fotograma dentro de los ficheros guardados se utiliza un nuevo fichero auxiliar de extensión .frames.mat que guarda la lista de fotogramas, los cuales son identificados por un número según el orden en el que aparecen en el vídeo, ordenada según el orden en el que se han guardado los datos de los fotogramas.
Gracias a la librería mmread se puede leer virtualmente cualquier archivo multimedia, utilizando AVbin y FFmpeg para la captura de datos.

4.1.1. Cambios en la funcionalidad

Lo primero que hay que destacar es que ahora es posible iniciar el programa con dos argumentos más. Este es un ejemplo de comando con los nuevos parámetros:

segmentation_ui(loaddir('./office_planta4', 'tilted_face.avi'), loadstrings('SmartRoom_objects.txt') , 120, 'video');
El tercer argumento es un entero que indica el número de imagen o fotograma por el que comenzar a anotar.
El cuarto argumento es un string que indica el modo en que se va a utilizar la herramienta. Solamente puede tener los valores:

- 'image': si se quiere utilizar el modo de anotar imágenes.
- 'video': si se quiere utilizar el modo de anotar vídeos.

En caso de que no se incluyan estos argumentos toman valores por defecto. El número de imagen o fotograma es 1 en ese caso, y el modo es el de anotar imágenes.
Una vez introducido el comando y abierto la nueva figura, se llaman a las mismas funciones que antes, sólo que realizan algo distinto cuando el modo introducido es el de anotar vídeos. A continuación se procede a explicar las nuevas funcionalidades de estas funciones en modo de vídeo.
En la función *segmentation_ui_OpeningFcn* se realiza la lectura del vídeo usando la librería externa mmread. Se realiza en esta función y no en *initialize_image* porque la lectura del vídeo es un proceso largo y costoso y no es eficiente que la herramienta lea todo el vídeo cada vez que cambie de fotograma. El vídeo leído se guarda en uno de los campos de la estructura de datos UI. Para ver los cambios realizados en esta función hay que observar la figura 22.

```
if length(varargin)==2
    i = 1;
    handles.mode = 'image';
    handles.labels = struct([]);
    handles.segmentation = struct([]);
    handles.polygons = struct([]);
elseif length(varargin)==4
    i = varargin{3};
    handles.mode = varargin{4};
    handles.labels = struct([]);
    handles.segmentation = struct([]);
    handles.polygons = struct([]);
    if strcmp(handles.mode, 'video')==true
        file = handles.images{1};
        handles.video = mmread(file);
    end
```

Figura 23: Cambios en segmentation_ui_OpeningFcn para la lectura de varios formatos de vídeo

En la función *initialize_image* se lee dentro del vídeo el fotograma específico que se va a anotar y se cargan los datos de los ficheros de manera diferente, ya que ahora se encontrarán las anotaciones de todos los fotogramas de un mismo vídeo en los mismos ficheros. Estos cambios se ven reflejados en la figura 23.

```
if strcmp(d.mode, 'video')==true
    file = d.images{1};

    d.image = d.video.frames(i).cdata;

    d.length = d.video.nrFramesTotal;

    [ly,lx,depth]=size(d.image);

    d.oldsegm     = d.segmentation;
    nuevo_frame=true;
    if exist([file '.frames.mat'],'file')
        loaded_frames=load([file '.frames.mat']);
        nf=loaded_frames.nf;
        loaded_segmentations= load_segmentation(file, ly, lx);
        for j=1:length(nf)
            if nf{j}==d.i
                pos=j;
                nuevo_frame=false;
            end
        end
        if nuevo_frame==false
            d.segmentation=loaded_segmentations(:,:,1,pos);
        else
            d.segmentation=zeros(ly,lx);
        end
    else
        d.segmentation= load_segmentation(file, ly, lx);
    end
    d.oldrefine   = 0;

    if exist([file '.labels.mat'],'file')
      if nuevo_frame==false
          loaded_labels=load([file '.labels.mat']);
          if length(d.labels)==0
              if pos<=length(loaded_labels.l)
                d.labels = loaded_labels.l{pos};
              end
          end
      end
    end
```

Figura 24: Cambios en la función initialize_image para la lectura de varios formatos de vídeo

En *showimage* se muestra el número de fotograma o imagen en el que se encuentra el usuario actualmente. Lo hace encima de la imagen. Este cambio en el código aparece en la figura 24.

```
if strcmp(d.mode, 'image')==true
    set(d.nImage,'String',['Image nº ',num2str( d.i )]);
elseif strcmp(d.mode, 'video')==true
    set(d.nImage,'String',['Frame nº ',num2str( d.i )]);
end
```

Figura 25: Cambios en la función showimage para la lectura de varios formatos de vídeo

Tras iniciar el programa en modo vídeo se muestra una interfaz como la mostrada en la ilustración 18.

Además la funcionalidad del botón save cambia en el modo de vídeo. Por la misma razón que cargar los datos se hace de forma diferente guardarlos se hace de forma distinta también. Para poder saber en qué orden están

guardados los datos de cada fotograma se usa un nuevo fichero de extensión .frames.mat que sirve de índice que muestra el orden en que se almacenan las anotaciones de cada fotograma. Los cambios de la función de activación llevados a cabo aparecen en la figura 25.

```
if strcmp(d.mode, 'image')==true
    filename = d.images{d.i};
    imwrite( s, [filename '.seg.gif']);
elseif strcmp(d.mode, 'video')==true
    filename = d.images{1};
    [ly,lx,depth]=size(s);
    segmentations = zeros(ly, lx, 1, 1);
    segmentations = im2uint8(segmentations);
    if exist([filename '.frames.mat'],'file')
        loaded_frames=load([filename '.frames.mat']);
        nf=loaded_frames.nf;
        loaded_segmentations = load_segmentation(filename, 1, 1);
        [ly,lx,x,depth]=size(loaded_segmentations);
        pos = depth+1;
        for j=1:length(nf)
            if nf{j}==nframe
                pos=j;
                nuevo_frame=false;
            end
        end
        for j=1:depth
            [map_aux,s_aux] = convert2_8bit(loaded_segmentations(:,:,1,j));
            segmentations(:,:,1,j) = s_aux;
        end
        segmentations(:,:,1,pos) = s;
    else
        nf=cell(1);
        segmentations(:,:,1,pos) = s;
    end
    imwrite( segmentations, [filename '.seg.gif']);
    if nuevo_frame == true
        nf{pos}=nframe;
        save([filename '.frames.mat'],'nf');
    end
end

...

if savelabels ~= 1
  warning('segmentation_ui:save_Callback','No labels saved!');
else
  if strcmp(d.mode, 'image')==true
     save([filename '.labels.mat'],'labels');
     buildSceneDescription([filename '.sceneDescription.txt'], [filename '.labels.mat'], [filename '.seg.gif']);
  elseif strcmp(d.mode, 'video')==true
     if exist([filename '.labels.mat'],'file')
       loaded_labels=load([filename '.labels.mat']);
       l=loaded_labels.l;
     else
       l=cell(1);
     end
     l{pos}=labels;
     save([filename '.labels.mat'],'l');
     buildSceneDescription([filename '.sceneDescription.txt'], [filename '.labels.mat'], [filename '.seg.gif'], [filename '.frames.mat']);
  end
end
```

Figura 26: Cambios en la función save_Callback para la lectura de varios formatos de vídeo

4.2. Adición de anotaciones automáticas

Para añadir la posibilidad de realizar anotaciones automáticas también se han utilizado dos detectores: un detector de caras y otro de personas. Ambos detectores utilizan el detector vision.CascadeObjectDetector de Matlab, el cual

detecta objetos utilizando algoritmos de Viola-Jones. El usuario tiene que elegir que detector quiere usar a través de la lista de detectores desplegable, y luego pulsar el botón apply. Tras esto, la aplicación abrirá un nuevo panel en el que se muestra la misma imagen o fotograma que se estaba tratando pero con los nuevos objetos detectados por el detector delimitados por regiones. El usuario tendrá la opción de cambiar el umbral a emplear en la detección de objetos, ya que el usado por defecto siempre será 4. Al lado del botón threshold aparecerá un consejo sobre qué valores escoger para el umbral para ese detector en concreto. Si se pulsa este botón aparecerá un campo con un valor de 4 por defecto, y que se podrá cambiar antes de pulsar 'accept' para cambiar el valor del umbral. En la propia imagen se podrán seleccionar las regiones que delimitan objetos que se correspondan con el tipo de objeto a detectar. Una vez hecho eso, el usuario podrá pulsar el botón finish para volver al panel anterior con las regiones seleccionadas como nuevas regiones etiquetadas.

4.2.1. Cambios en la funcionalidad

Como se puede ver en la ilustración 18, ahora hay una lista desplegable de detectores de los cuales se puede elegir uno antes de pulsar el botón apply y pasar a la interfaz de detección. Este botón pasa como argumentos los nombres de las imágenes o vídeos, las etiquetas que se pueden asignar, el número de imagen o fotograma actual, el detector seleccionado, el modo usado y el vídeo leído en caso de estar en modo vídeo a la interfaz de detección. Ésta abre la figura detection_ui.fig y llama a funciones del fichero detection_ui.m similares a las de segmentation_ui.m.
La función *detection_ui_OpeningFcn* lee los parámetros y los guarda en la estructura de datos UI, y pone otros valores por defecto en la estructura, como el umbral de detección a 4. Por último llama a *det_initialize_image*. El código de la función es mostrado en la figura 26.

```
function detection_ui_OpeningFcn(hObject, eventdata, handles, varargin)

handles.images     = varargin{1}; % first  parameter: images
handles.categories = varargin{2}; % second parameter: categories
i = varargin{3};
handles.detector = varargin{4};
handles.mode = varargin{5};
if strcmp(handles.mode, 'video')==true
    handles.video = varargin{6};
end
handles.colormap = randomcolormap();
handles.oldsegm = 0;
handles.output = hObject; % Choose default command line output for segmentation_ui
handles.MergeThreshold = 4;
guidata(hObject, handles); % Update handles structure

det_initialize_image(hObject,i);
```

Figura 27: Función detection_ui_OpeningFcn

En la función *det_initialize_image* se cargan los datos de los ficheros auxiliares .mat igual que en segmentation_ui.m, pero la imagen GIF que guarda la segmentación no la usa, ya que parte de una matriz de segmentación inicializada a ceros. También pone la matriz de regiones seleccionadas a ceros. Después se dispone a aplicar el detector recibido por parámetro. Al aplicar el detector de la librería de vision se reciben los puntos de los polígonos que delimitan los objetos detectados, y se utilizan para cambiar la segmentación e incluir las nuevas regiones en ella. Lo último que hace es llamar a la función *det_showimage*. Para ver lo que cambia *det_initialize_image* con respecto a *initialize_image* de segmentation_ui.m hay que observar la figura 27.

```
[ly,lx,depth]=size(d.image);

d.segmentation = zeros(ly, lx);

if d.detector == 1
    detector = vision.CascadeObjectDetector();
end
if d.detector == 2
    detector = vision.CascadeObjectDetector('UpperBody');
end

detector.MergeThreshold = d.MergeThreshold;

bboxes = step(detector, d.image);

[n_obj,bx,depth]=size(bboxes);

X=ones(ly,1)*(1:lx);
Y=(1:ly).'*ones(1,lx);

for j=1:n_obj
    px(1) = bboxes(j);
    py(1) = bboxes(j+n_obj);
    px(2) = bboxes(j) + bboxes(j+2*n_obj);
    py(2) = bboxes(j+n_obj);
    px(3) = bboxes(j) + bboxes(j+2*n_obj);
    py(3) = bboxes(j+n_obj) + bboxes(j+3*n_obj);
    px(4) = bboxes(j);
    py(4) = bboxes(j+n_obj) + bboxes(j+3*n_obj);
    in=inpolygon(X,Y,px,py);
    d.oldsegm=d.segmentation;
    token=det_findunusedtoken(d.segmentation);
    d.segmentation(in)=token;
    i=length(d.polygons)+1;
    d.polygons(i).segmenttoken=token;
    d.polygons(i).px=px;
    d.polygons(i).py=py;
end

guidata(handle,d);

det_showimage(d);
```

Figura 28: Cambios de det_initialize_image con respecto a initialize_image

Esta función, *det_showimage*, hace lo mismo que su homóloga en segmentation_ui.m, sólo que muestra la segmentación de la imagen siempre, sin depender de ningún checkbox, y cambia el consejo que da para poner valores al umbral en función del detector seleccionado.
La figura 28 enseña el código de la función *det_showimage*.

```
function det_showimage(d)


  temp=visualize_segm2(d.image,d.segmentation);


sel=(d.selstatus==1);

temp(sel) = .3.*temp(sel)+.7.*256.*d.red(sel);
axes(d.img);
h=image(temp);
    %h=imshow(temp);
set(h,'ButtonDownFcn',@det_img_ButtonDownFcn);
if d.detector == 1
    set(d.tip,'String',['Use values between 0 and 15 for the threshold']);
elseif d.detector == 2
    set(d.tip,'String',['Use values between 0 and 20 for the threshold']);
end
```

Figura 29: Función det_showimage

Tras esto se muestra una interfaz como la de la ilustración 19.

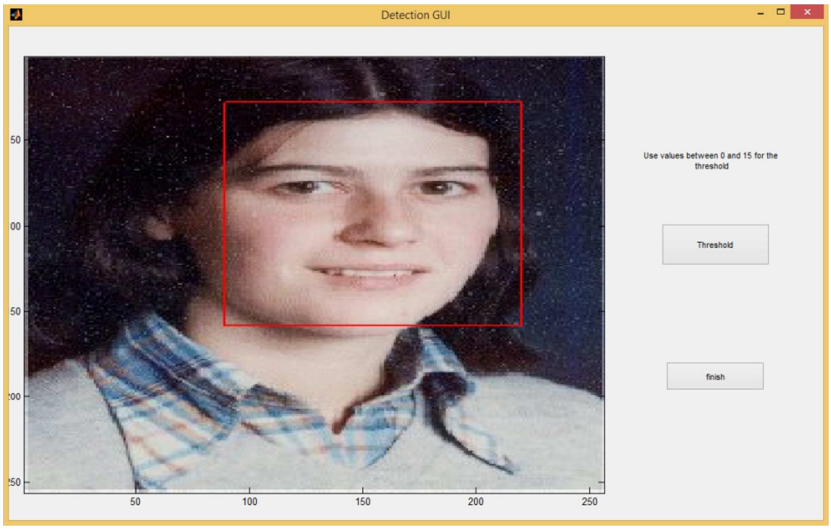

Ilustración 19: Interfaz de detección de la herramienta de anotación final

En esta ilustración el usuario ha seleccionado las regiones que considera que se ajustan a los objetos que el detector está intentando detectar. Para seleccionar regiones se utiliza la misma funcionalidad que en la interfaz de segmentación.
El usuario también puede pulsar el botón threshold si quiere ajustar el umbral utilizado por el detector a la hora de considerar a un objeto como el objeto a detectar o no. Si lo sube se detectarán menos objetos y se podrán eliminar algunos falsos positivos. Al pulsarle el botón se abrirá una ventana

emergente en la que introducir un valor para el umbral. Ya aparecerá como valor por defecto un 4, antes de cambiarse. Si se pulsa ‘accept’ se guarda en la estructura de datos UI el nuevo valor de umbral y se vuelve a aplicar el detector sobre la imagen con el umbral cambiado al llamar otra vez a la función *det_initialize_image*. La función de activación de este botón es mostrada en la figura 29.

```
function Threshold_Callback(hObject, eventdata, d)

answer=inputdlg({'Enter detection threshold:'},'Threshold',1,{'4'});
if numel(answer) ~= 1, return, end  % user clicked 'cancel'

k=str2num(answer{1});

d.MergeThreshold = k;

guidata(hObject,d);

det_initialize_image(hObject,d.i);
```

Figura 30: Función Threshold_Callback

Una vez que se hayan conseguido seleccionar las regiones de interés, se puede pulsar el botón finish para volver a la interfaz de segmentación con las nuevas regiones etiquetadas en la imagen o fotograma. Esto lo hace reuniendo todas las ids de regiones seleccionadas y añadiendo a la lista de etiquetas asignadas las nuevas regiones con su etiqueta asignada. Esta será para todas la misma, la que corresponde con el objeto a detectar. Después vuelve a la interfaz de segmentación pasando como parámetros, además de los cuatro primeros ya comentados, las etiquetas asignadas a las regiones, la segmentación de la imagen y el vídeo leído si la herramienta está en modo vídeo. También cierra la interfaz de detección. La figura 30 muestra la función que se ejecuta al pulsar el botón.

```
function finish_Callback(hObject, eventdata, d)

selected = d.segmentation(d.selstatus == 1);

if length(selected) > 0
    segments=deleteduplicates(selected);
    if numel(segments)<1, close(d.figure1), end

    if d.detector == 1
        object = 'Object\AnimatedObject\Face';
    end
    if d.detector == 2
        object = 'Object\AnimatedObject\Person';
    end

    k=length(d.usefulpolygons)+1;

    for s=1:length(segments) % for all selected segments

      i=1;
      while i<=length(d.labels) % already an entry existing?
        if d.labels(i).segmenttoken==segments(s), break, end
        i=i+1;
      end % if loop runs through, i is automatically pointing to next entry

      d.labels(i).segmenttoken = segments(s);

      d.labels(i).label        = object;
```

…

```
    end

    guidata(hObject,d);

    if strcmp(d.mode, 'image')==true
        segmentation_ui (d.images, d.categories, d.i, d.mode, d.labels, d.segmentation, d.usefulpolygons)
    elseif strcmp(d.mode, 'video')==true
        segmentation_ui (d.images, d.categories, d.i, d.mode, d.labels, d.segmentation, d.usefulpolygons, d.video)
    end
else
    if strcmp(d.mode, 'image')==true
        segmentation_ui (d.images, d.categories, d.i, d.mode)
    elseif strcmp(d.mode, 'video')==true
        segmentation_ui (d.images, d.categories, d.i, d.mode, d.video)
    end
end

close(d.figure1);
```

Figura 31: Función finish_Callback

El vídeo leído es guardado en la estructura de datos UI en *segmentation_ui_OpeningFcn*. La lista de etiquetas asignadas a regiones se guarda en la estructura en la función *segmentation_ui_OpeningFcn*, en lugar de cargarla del fichero de extensión .labels.mat.
Puede verse en los cambios a la función mostrados en la figura 31.

```
elseif length(varargin)==5
    i = varargin{3};
    handles.mode = varargin{4};
    handles.labels = struct([]);
    handles.segmentation = struct([]);
    handles.polygons = struct([]);
    handles.video = varargin{5};
elseif length(varargin)==7
    i = varargin{3};
    handles.mode = varargin{4};
    handles.labels = varargin{5};
    handles.segmentation = varargin{6};
    handles.polygons = varargin{7};
    if strcmp(handles.mode, 'video')==true
        file = handles.images{1};
        handles.video = mmread(file);
    end
elseif length(varargin)==8
    i = varargin{3};
    handles.mode = varargin{4};
    handles.labels = varargin{5};
    handles.segmentation = varargin{6};
    handles.polygons = varargin{7};
    handles.video = varargin{8};
else
    close(handles.figure1);
    return
end
```

Figura 32: Cambios en la función segmentation_ui_OpeningFcn para la adición de anotaciones automáticas

Y, en cuanto a la segmentación, se carga la segmentación de la imagen GIF como siempre en *initialize_image*, pero además se modifica la segmentación para incluir las nuevas regiones delimitadas en la detección. En la figura 32 se puede ver como se añaden las nuevas regiones, las cuales están guardadas en el campo oldsegm, a la segmentación.

```
if length(d.oldsegm) ~= 0
    for i=1:length(d.labels)
        segmenttoken = d.labels(i).segmenttoken;
        for j=1:numel(d.segmentation)
            if d.oldsegm(j) == segmenttoken
                d.segmentation(j) = segmenttoken;
            end
        end
    end
end
```

Figura 33: Cambios en la función initialize_image para la adición de anotaciones automáticas

4.3. Implementación de mecanismo de propagación de anotaciones

El mecanismo de propagación de anotaciones implementado es aquel en el que el usuario va arrastrando la región etiquetada mientras cambia de fotogramas dentro de un mismo vídeo, y va ajustando la región a medida que el objeto al cual delimita va cambiando de posición y/o tamaño. Para utilizarlo hay que pulsar el botón start edit mode, tras lo cual se entra en un modo en el que se pueden arrastrar todas las regiones de la imagen o fotograma para cambiarlos de posición y/o tamaño. Después de esto, hay que mover la región que se desea propagar, ya que la última región movida es la que se va a propagar al siguiente fotograma. Acto seguido se pulsa el botón next o prev todas las veces que se quiera a la vez que se va ajustando la región si es preciso. Cuando la aplicación está en “edit mode” los botones next y prev guardan antes de cambiar de fotograma sin preguntárselo al usuario. Cuando se quiera terminar de propagar anotaciones se pulsa el botón finish edit mode, el cual ha remplazado al de start edit mode.

4.3.1. Cambios en la funcionalidad

Se ha añadido un “edit mode”, el cual es un modo que permite ajustar la segmentación en la imagen, cambiando de posición tanto el polígono en sí como sus puntos. Este modo está inicialmente desactivado. Se pone a ‘false’ un campo de la estructura de datos UI en la función inicial *segmentation_ui_OpeningFcn*. Pero cuando se pulsa el botón start edit mode que puede verse en la ilustración 18, la herramienta entra en este modo. Este botón tiene una función de activación que aparece en la figura 33.

```
function editmode_Callback(hObject, eventdata, d)

if d.edit==false
    d.edit=true;
    guidata(hObject,d);
    showimage(d);
elseif d.edit==true
    d.edit=false;
end

if d.edit==true
    d.lastposition = struct([]);
    initialize_draggable (hObject,d);
elseif d.edit==false
    change_segmentation (hObject,d);
    d = guidata(gcbo);
    showimage(d);
end
```

Figura 34: Función editmode_Callback

Las regiones pasan a ser arrastrables, y esto es gracias a una nueva lista que guarda para cada polígono que se encuentra en la imagen sus puntos que lo delimitan. De esa forma pueden cambiarse los puntos de posición. Las regiones pasan a ser arrastrables tras llamarse a la función *initialize_draggable*, que aparece en la figura 34.

```
function initialize_draggable (hObject,d)

d.draggable=cell(1);
for i=1:length(d.polygons)
    position=zeros(length(d.polygons(i).px), 2);
    for j=1:length(d.polygons(i).px)
        position(j)=d.polygons(i).px(j);
        position(j+length(d.polygons(i).px))=d.polygons(i).py(j);
    end
    d.draggable{i} = impoly(gca, position);
    setColor(d.draggable{i},'red');
    fcn = makeConstrainToRectFcn('impoly',get(gca,'XLim'),...
        get(gca,'YLim'));
    setPositionConstraintFcn(d.draggable{i},fcn);
    addNewPositionCallback(d.draggable{i},@last_draggable);
end
guidata(hObject,d);
```

Figura 35: Función initialize_draggable

Esta nueva lista se guarda también al pulsar el botón save. Se guarda en un fichero de extensión .polygons.mat de igual manera que la lista de etiquetas asignadas se guarda en el fichero de extensión .labels.mat. Esto puede verse en el código de la figura 35.

```
j=1;
savepolygons=0;
for l=1:length(d.polygons) % only save non-empty tokens! (use for display, too?!)
  mypolygon=d.polygons(l).segmenttoken;
  if( numel(find(d.segmentation==mypolygon,1)) )>0
    savepolygons=1;
    polygons(j)=d.polygons(l);
    polygons(j).segmenttoken=map( polygons(j).segmenttoken+1 ); % and, very important: map to uint8 values!
    j=j+1;
  end
end

if savepolygons ~= 1
  warning('segmentation_ui:save_Callback','No polygons saved!');
else
    if strcmp(d.mode, 'image')==true
        save([filename '.polygons.mat'],'polygons');
    elseif strcmp(d.mode, 'video')==true
        if exist([filename '.polygons.mat'],'file')
            loaded_polygons=load([filename '.polygons.mat']);
            p=loaded_polygons.p;
        else
            p=cell(1);
        end
        p{pos}=polygons;
        save([filename '.polygons.mat'],'p');
    end
end
```

Figura 36: Cambios en la función save_Callback para la implementación de mecanismo de propagación de anotaciones

También se carga la lista de polígonos igual que la de etiquetas en la función *initialize_image*, como puede verse en la figura 36.

```
if exist([file '.polygons.mat'],'file')
  polygons=load([file '.polygons.mat']);
  if length(d.polygons)==0
      d.polygons = polygons.polygons;
  end
end
...

if exist([file '.polygons.mat'],'file')
  if nuevo_frame==false
      loaded_polygons=load([file '.polygons.mat']);
      if length(d.polygons)==0
          if pos<=length(loaded_polygons.p)
            d.polygons = loaded_polygons.p{pos};
          end
      end
  end
end
```

Figura 37: Cambios en la función initialize_image para la implementación de mecanismo de propagación de anotaciones

Y, por último, también se reciben los polígonos de la interfaz de detección en la de segmentación de la misma manera que las etiquetas, además sólo se reciben los polígonos que han sido etiquetados. El cambio se encuentra en la función de activación del botón finish, y se muestra en la figura 37.

```
  j=1;
  while j<=length(d.polygons)
     if d.polygons(j).segmenttoken == d.labels(i).segmenttoken
        d.usefulpolygons(k).px = d.polygons(j).px;
        d.usefulpolygons(k).py = d.polygons(j).py;
        break;
     end
     j=j+1;
  end
  d.usefulpolygons(k).segmenttoken = d.labels(i).segmenttoken;
  k = k+1;
end

guidata(hObject,d);

if strcmp(d.mode, 'image')==true
    segmentation_ui (d.images, d.categories, d.i, d.mode, d.labels, d.segmentation, d.usefulpolygons)
elseif strcmp(d.mode, 'video')==true
    segmentation_ui (d.images, d.categories, d.i, d.mode, d.labels, d.segmentation, d.usefulpolygons, d.video)
end
```

Figura 38: Cambios en la función finish_Callback para la implementación de mecanismo de propagación de anotaciones

Esta lista va aumentando cada vez que se pulsa el botón polygon para crear un polígono y cada vez que se devuelve a la interfaz de segmentación una o varias regiones seleccionadas en la de detección. El cambio a la función que se ejecuta tras pulsar el botón polygon aparece mostrado en la figura 38.

```
function polygon_Callback(hObject, eventdata, d)

[ly,lx,depth]=size(d.image);

[px,py]=getline(d.img);
X=ones(ly,1)*(1:lx);
Y=(1:ly).'*ones(1,lx);
in=inpolygon(X,Y,px,py);

d.oldsegm=d.segmentation;
token=findunusedtoken(d.segmentation);
d.segmentation(in)=token;
i=length(d.polygons)+1;
d.polygons(i).segmenttoken=token;
d.polygons(i).px=px;
d.polygons(i).py=py;
guidata(hObject,d);

showimage(d);
```

Figura 39: Cambios en la función polygon_Callback para la implementación de mecanismo de propagación de anotaciones

Cuando se pulsa el botón finish edit mode, que ocupa el lugar del anterior botón start edit mode, se abandona este modo y se cambia la segmentación y la lista de los polígonos para actualizarlos con las nuevas posiciones de las regiones. Esto lo hace la función *change_segmentation*, cuyo código es mostrado en la figura 39.

```
function change_segmentation (hObject,d)

[ly,lx,depth]=size(d.image);
d.segmentation = zeros(ly, lx);
d.selstatus = zeros(ly, lx);
X=ones(ly,1)*(1:lx);
Y=(1:ly).'*ones(1,lx);
d.lastlabel = struct([]);
for i=1:length(d.polygons)
    position = getPosition(d.draggable{i});
    if length(d.lastposition) ~= 0
        if d.lastposition==position
            for j=1:length(d.labels)
                if d.labels(j).segmenttoken == d.polygons(i).segmenttoken
                    d.lastlabel = d.labels(j).label;
                end
            end
        end
    end
    for j=1:length(d.polygons(i).px)
        d.polygons(i).px(j)=position(j);
        d.polygons(i).py(j)=position(j+length(d.polygons(i).px));
    end
    in=inpolygon(X,Y,d.polygons(i).px,d.polygons(i).py);
    d.segmentation(in)=d.polygons(i).segmenttoken;
end
guidata(hObject,d);
```

Figura 40: Función change_segmentation

Además, durante el “edit mode” se guarda la posición del último polígono ajustado, para así poder propagarlo a la imagen siguiente. Ocurre gracias a la función *last_draggable*, la cual es llamada cada vez que cambia un arrastrable de posición. Su código aparece en la figura 40.

```
function last_draggable(p)

d = guidata(gcbo);

d.lastposition = p;

guidata(gcbo,d);
```

Figura 41: Función last_draggable

Al pulsar los botones next o prev en este modo no se pregunta al usuario si quiere guardar, sino que se guarda directamente, tras actualizar la segmentación y la lista de polígonos. Es una medida para realizar la propagación de anotación más rápida. En la figura 41 puede verse el cambio a la función de activación del botón next, el cual es el mismo que el de la del botón prev también.

```
function next_Callback(hObject, eventdata, handles)

d = guidata(gcbo);

if d.edit==false
    answer=questdlg('save before going to next image?','save?');
    if strcmp(answer,'Cancel'), return, end
    if strcmp(answer,'Yes'), save_Callback(handles.save,[],handles); end
elseif d.edit==true
    change_segmentation (hObject,d);
    d = guidata(gcbo);
    save_Callback(d.save,[],d);
end

i = d.i + 1;
if( i > d.length )
  msgbox('End of list reached! You may close the application.', ...
                'End of list','none','modal');

               %% delete all the segmentation !!rely on the fact the user did
               %% NOT name his/her files with *.segpre*.ppm & name them "name" . "ext"!!

   for i=1:length(d.images)
     file = d.images{i};
     delete([ file '.segpre*.ppm']);
   end

  return
end

d.labels = struct([]);
d.polygons = struct([]);
d.segmentation = struct([]);
guidata(hObject,d);
initialize_image(gcbo,i);

if d.edit==true
    d = guidata(gcbo);
    initialize_draggable (hObject,d)
end
```

Figura 42: Cambios en la función next_Callback para la implementación de mecanismo de propagación de anotaciones

También se ha hecho que pulsar la tecla de la flecha a la derecha sea equivalente a pulsar el botón next, y que la tecla de la flecha a la izquierda lo sea al botón prev. Para ello se ha implementado una función cuyo código se ve en la figura 42.

```
function figure1_KeyPressFcn(hObject, eventdata, handles)
if strcmp (eventdata.Key, 'rightarrow') == true
    next_Callback(hObject, eventdata, handles)
elseif strcmp (eventdata.Key, 'leftarrow') == true
    prev_Callback(hObject, eventdata, handles)
end
```

Figura 43: Función figure1_KeyPressFcn

Al inicializar la imagen o fotograma nuevo en “edit mode” con *initialize_image* se crea un nuevo polígono con la misma posición que el último polígono ajustado y con su etiqueta asignada, si es que tenía, y cambia la segmentación para incluir esta nueva región. Este cambio se ve reflejado en la figura 43.

```
if d.edit==true
    if length(d.lastposition) ~= 0
        X=ones(ly,1)*(1:lx);
        Y=(1:ly).'*ones(1,lx);
        pos = length(d.polygons)+1;
        [ly,lx,depth]=size(d.lastposition);
        d.polygons(pos).px = zeros(ly, 1);
        d.polygons(pos).py = zeros(ly, 1);
        for j=1:ly
            d.polygons(pos).px(j)=d.lastposition(j);
            d.polygons(pos).py(j)=d.lastposition(j+ly);
        end
        in=inpolygon(X,Y,d.polygons(pos).px,d.polygons(pos).py);
        d.oldsegm=d.segmentation;
        token=findunusedtoken(d.segmentation);
        d.segmentation(in)=token;
        d.polygons(pos).segmenttoken=token;
        if length(d.lastlabel) ~= 0
            pos = length(d.labels)+1;
            d.labels(pos).segmenttoken=token;
            d.labels(pos).label=d.lastlabel;
        end
    end
end
```

Figura 44: Más cambios en la función initialize_image para la implementación de mecanismo de propagación de anotaciones

También, como sigue en este modo todavía, las regiones de esta imagen, incluyendo la propagada, son arrastrables hasta que se pulse el botón finish edit mode.

En la figura 45 se muestra el diagrama de bloques de esta herramienta, el cual muestra sólo las nuevas funcionalidades con respecto a la inicial.

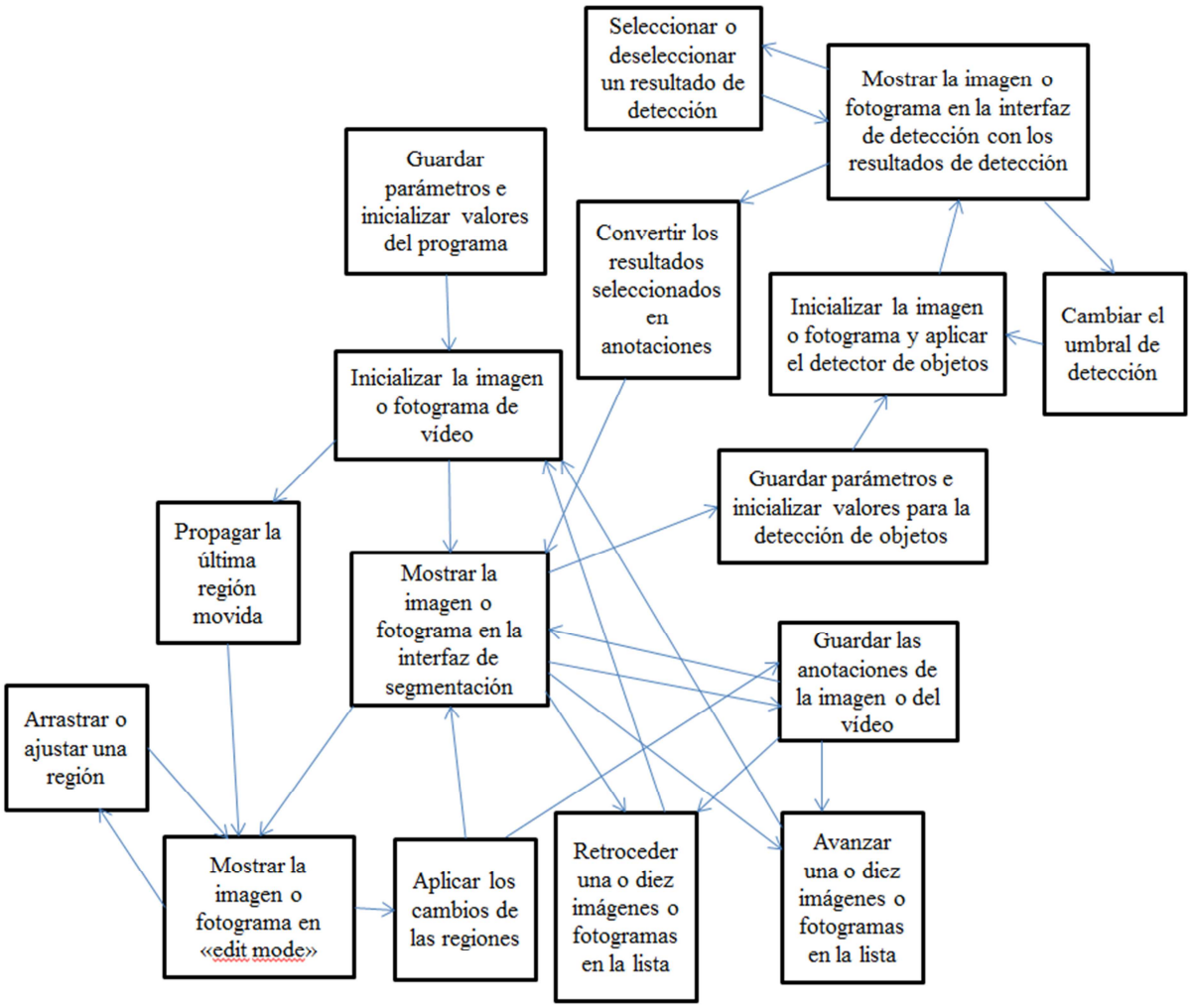

Figura 45: Diagrama de bloques del sistema de anotación final

5. Pruebas y validación

5.1. Pruebas de funcionalidad

Para realizar las pruebas de funcionalidad [6] se van a probar los siguientes requisitos funcionales:

- RF1: Permitir abrir una o varias imágenes para anotarlas al igual que lo hacía el sistema inicial.
- RF2: Ser capaz de leer vídeos en cualquier formato y navegar entre sus fotogramas sin problemas.
- RF3: Poder guardar las anotaciones realizadas en distintos fotogramas de un vídeo y luego cargarlas al tal cual se guardaron.
- RF4: Tener la posibilidad de aplicar un detector de objetos a la imagen o fotograma actual, seleccionar varios de los objetos detectados y que éstos acaben formando parte de las anotaciones de la imagen o vídeo.
- RF5: Poder cambiar el umbral del detector de objetos de manera que cambien los resultados mostrados por el detector.
- RF6: Permitir mover las regiones y ajustarlas.
- RF7: Ser capaz de propagar una región de una imagen o fotograma a otro de manera que se vaya guardando junto con su etiqueta asignada a medida que se va cambiando de imagen o fotograma y se va ajustando la región.

Cada prueba de funcionalidad va a comprobar que se cumple uno de los requisitos funcionales. Cada una va a mostrar el procedimiento que ha llevado a cabo y el resultado obtenido.

5.1.1. Prueba 01

Es la prueba que comprueba que se cumple el RF1.

5.1.1.1. Procedimiento

Se utiliza el comando:
segmentation_ui(loaddir('./office_planta4', '.jpg'), loadstrings('SmartRoom_objects.txt'));*
Solamente tiene dos argumentos, igual que los comandos utilizados para la herramienta de anotación inicial.

Se abre la primera imagen correctamente, se delimitan regiones con polígonos, se asignan etiquetas a estas regiones y se guarda. Este paso del procedimiento se puede ver en la ilustración 20.
Se pasa a la siguiente imagen y se abre correctamente también.
Se vuelve a la primera imagen y las anotaciones guardadas se han cargado correctamente. Se vuelve a ver todo exactamente como en la ilustración 20 otra vez.

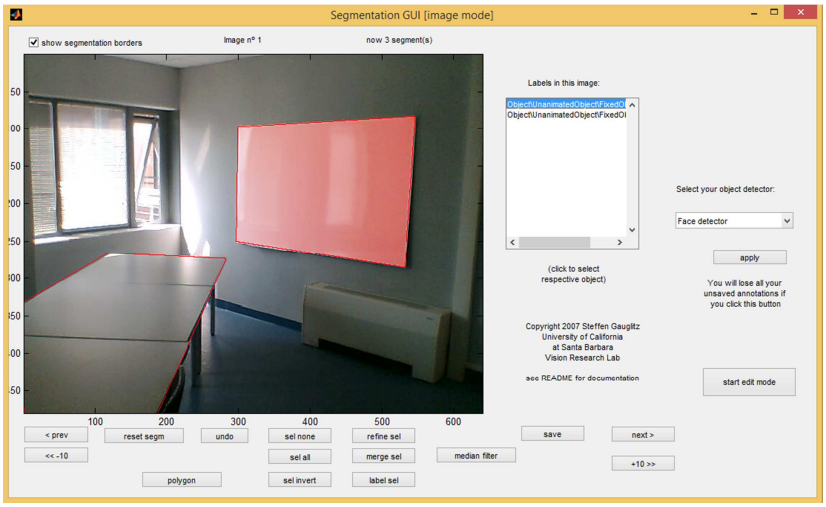

Ilustración 20: Prueba 01 regiones delimitadas y etiquetadas en imagen

5.1.1.2. Resultado

Ninguna de las mejoras añadidas ha perjudicado las funcionalidades del sistema de anotación inicial.

5.1.2. Prueba 02

Es la que se asegura de que el RF2 y el RF3 se cumplen.

5.1.2.1. Procedimiento

Se utiliza el comando:
segmentation_ui(loaddir('./office_planta4', 'tud-campus-sequence-gt.avi'), loadstrings('SmartRoom_objects.txt') , 12, 'video');
El cual abre un video en un formato de vídeo comprimido, aunque tenga la extensión .avi.
Se abre el fotograma número 12 adecuadamente, se delimitan regiones con polígonos, se asignan etiquetas a estas regiones y se guarda, tal y como se observa en la ilustración 21.

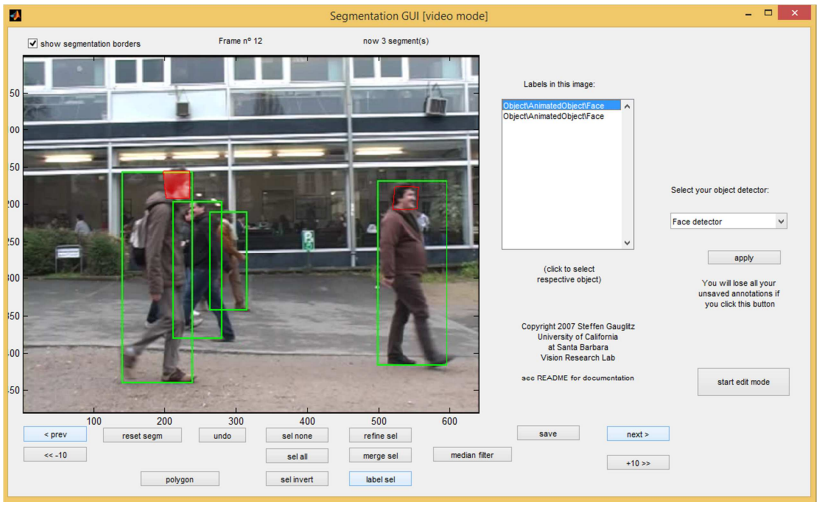

Ilustración 21: Prueba 02 regiones delimitadas y etiquetadas en primer fotograma

Se pasa al siguiente fotograma, el número 13, se ve que cambia ligeramente la imagen y se hace lo mismo. En la ilustración 22 se ve cómo se lleva a cabo este paso.

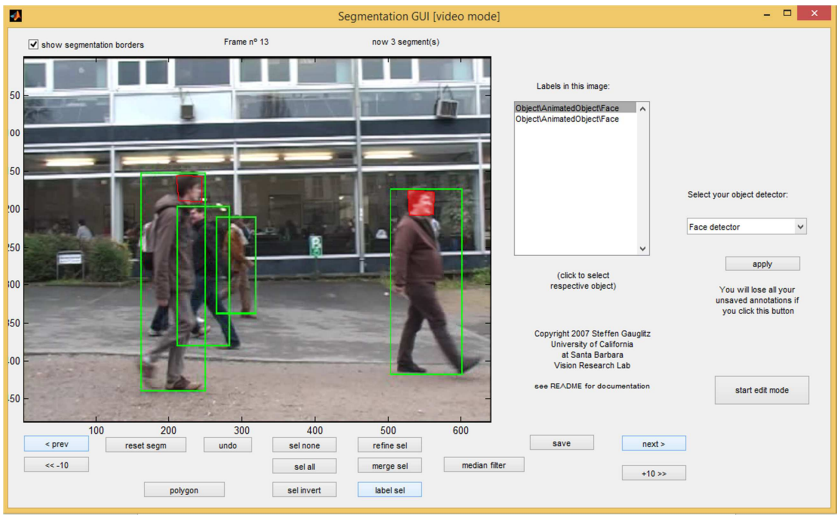

Ilustración 22: Prueba 02 regiones delimitadas y etiquetadas en segundo fotograma

Se pasa al siguiente fotograma, el número 14, se ve que la imagen cambia ligeramente también, de manera que todos o casi todos los objetos en movimiento se han desplazado en la misma dirección que en el anterior cambio. Así se puede ver en la ilustración 23.

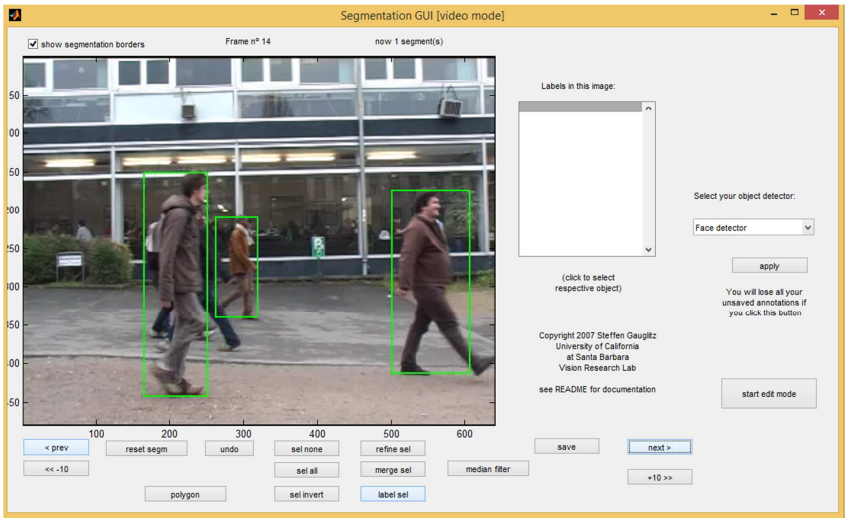

Ilustración 23: Prueba 02 tercer fotograma

Se pasa al fotograma anterior, el número 13, y resulta ser el fotograma abierto antes del 14. Las anotaciones guardadas se han cargado correctamente, lo que quiere decir que se vuelve a ver lo mismo que en la ilustración 22.
Se pasa al fotograma anterior, el número 12, y resulta ser el primer fotograma abierto. Las anotaciones se cargan adecuadamente también. Todo ha quedado igual que como se ve en la ilustración 21.

5.1.2.2. Resultado

Se puede leer un formato que no es muy común y se puede navegar entre los fotogramas hacia adelante y hacia atrás adecuadamente. A pesar de que en los vídeos se guardan los datos de todos los fotogramas en los mismos ficheros, se consiguen guardar y cargar las anotaciones adecuadamente.

5.1.3. Prueba 03

Se encarga de demostrar que el RF4 y el RF5 son de los requisitos que se cumplen.

5.1.3.1. Procedimiento

Se utiliza el comando:
segmentation_ui(loaddir('./office_planta4', 'visionteam.jpg'), loadstrings('SmartRoom_objects.txt'));
Se abre la imagen, se selecciona el detector de personas y se aplica.

Se abre una interfaz de detección que muestra los resultados de aplicar un detector con un umbral muy bajo, es decir, que detecta objetos que no son personas. Este paso del procedimiento realizado puede verse en la ilustración 24.

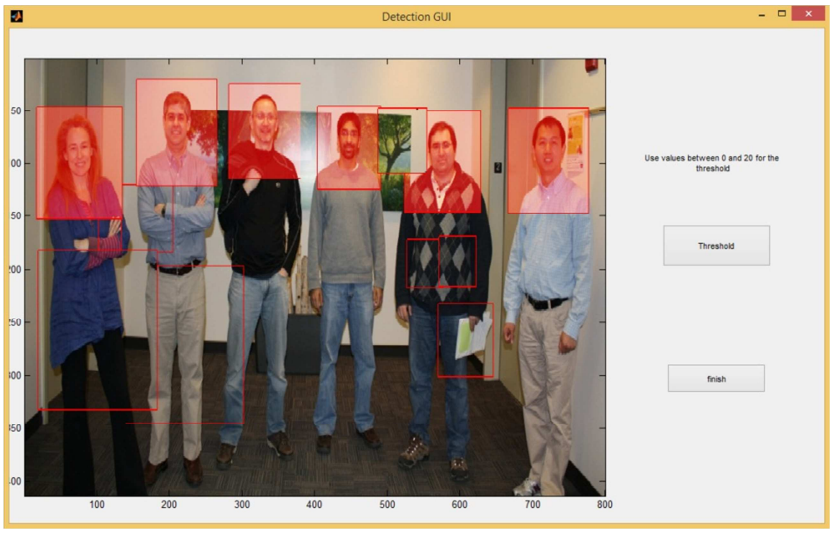

Ilustración 24: Prueba 03 detección con un umbral bajo

Se cambia el umbral a uno más alto que no detecte objetos que no sean personas.
Se seleccionan todas las regiones. Este paso aparece mostrado en la ilustración 25.

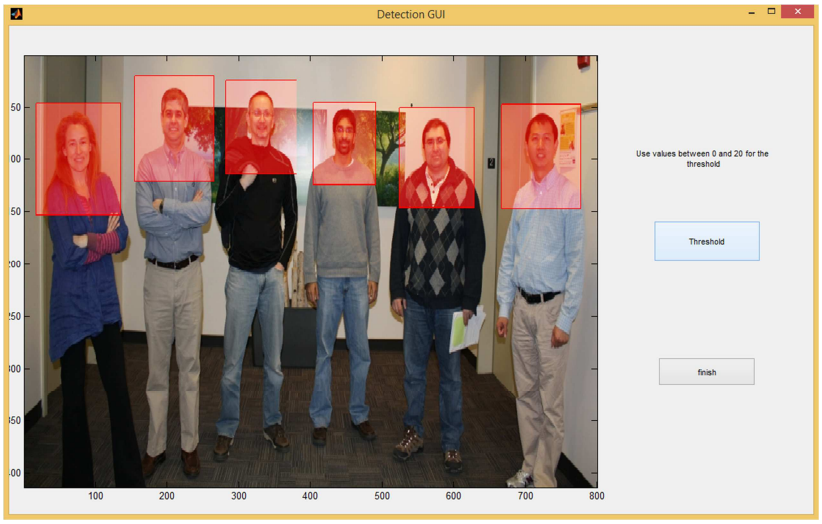

Ilustración 25: Prueba 03 detección con un umbral alto

Se vuelve a la interfaz de segmentación y todas las regiones seleccionadas ahora son regiones etiquetadas como personas. Este paso final es mostrado en la ilustración 26.

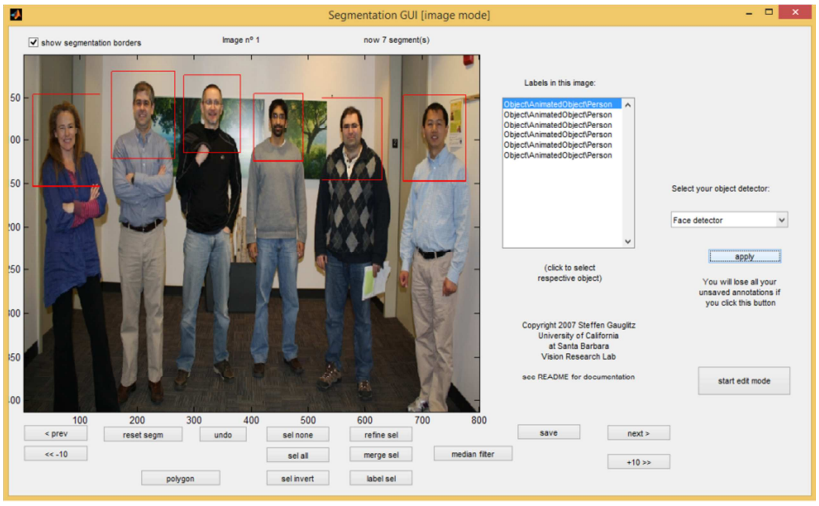

Ilustración 26: Prueba 03 anotaciones automáticas realizadas

5.1.3.2. Resultado

Las anotaciones automáticas funcionan correctamente. Tras utilizar el detector y seleccionar los resultados válidos se han mostrado las nuevas regiones etiquetadas correctamente. También se ha demostrado que cambiar el umbral del detector influye en los resultados que proporciona.

5.1.4. Prueba 04

Es la que se utiliza para demostrar que el RF6 se cumple.

5.1.4.1. Procedimiento

Se utiliza el comando:
segmentation_ui(loaddir('./office_planta4', 'clean_scene_002.jpg'), loadstrings('SmartRoom_objects.txt'));
Se abre la imagen, se delimita al azar una región de cuatro puntos manualmente y se la etiqueta como una puerta. En la ilustración 27 se puede ver este comienzo.

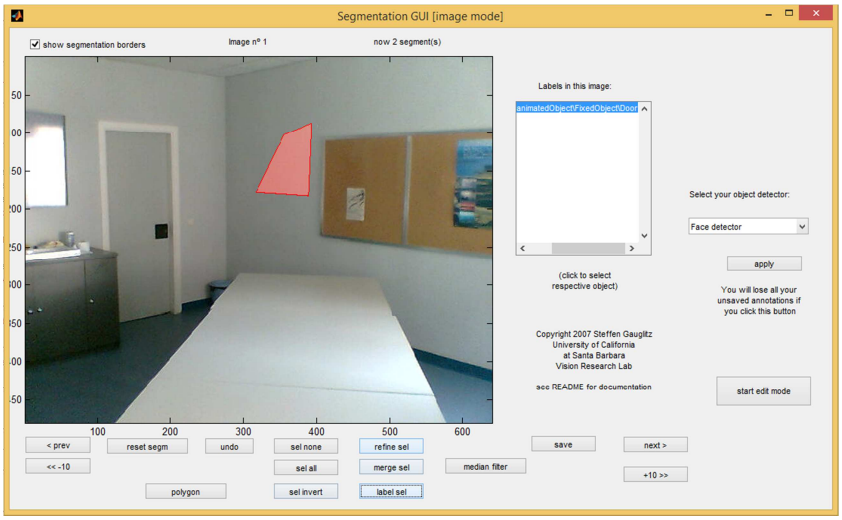

Ilustración 27: Prueba 04 delimitar una región al azar

Se entra en “edit mode” y la región pasa a ser arrastrable. Este paso puede verse en la ilustración 28.

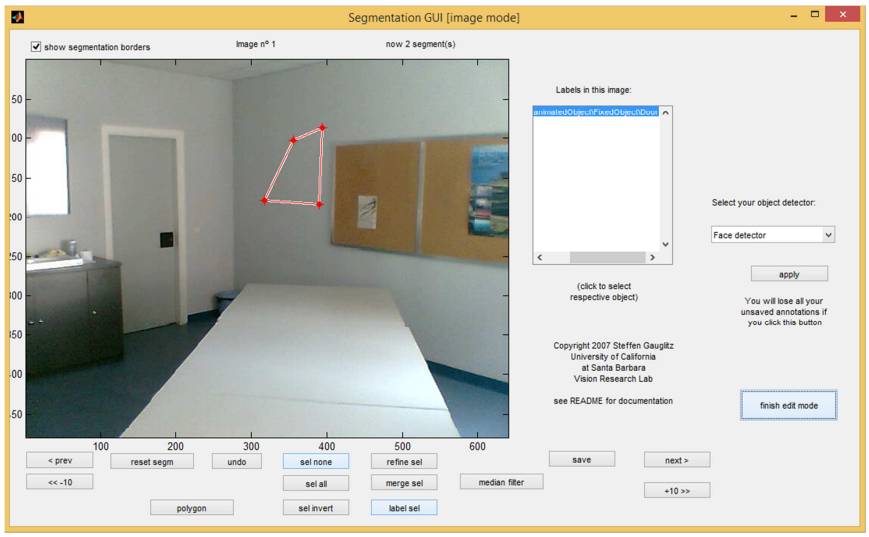

Ilustración 28: Prueba 04 entrar en “edit mode”

Se mueve la región para que cuadre con la puerta. Así lo muestra la ilustración 29.

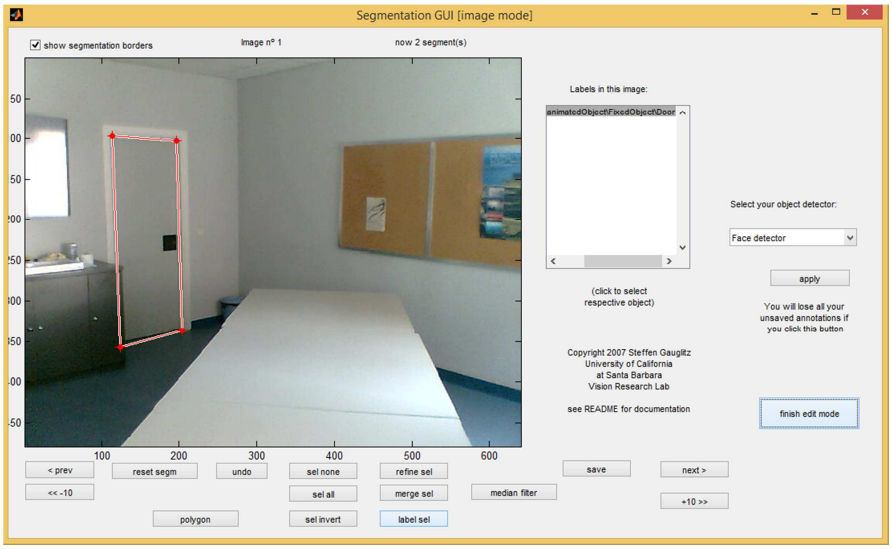

Ilustración 29: Prueba 04 ajustar la región

Se sale de “edit mode” y la región ha cambiado a su nueva posición adecuadamente, tal y como se puede observarse en la ilustración 30.

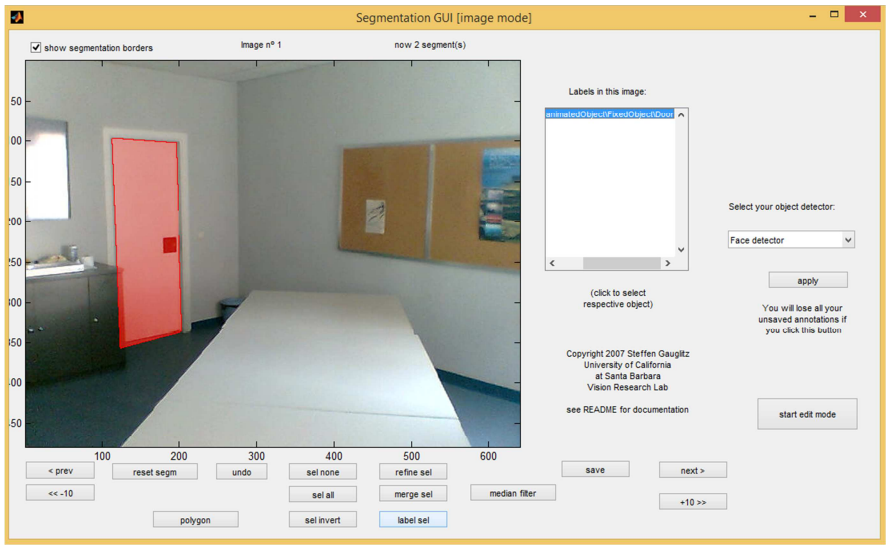

Ilustración 30: Prueba 04 región en diferente posición y con diferente tamaño

5.1.4.2. Resultado

Se pueden ajustar las regiones dentro de una imagen o fotograma sin ningún problema.

5.1.5. Prueba 05

Comprueba el RF7 para ver si se cumple.

5.1.5.1. Procedimiento

Se utiliza el comando:
segmentation_ui(loaddir('./office_planta4', 'tilted_face.avi'), loadstrings('SmartRoom_objects.txt') , 1, 'video');
Se abre el primer fotograma, se utiliza el detector de caras para anotar una cara automáticamente. Este paso inicial aparece en la ilustración 31.

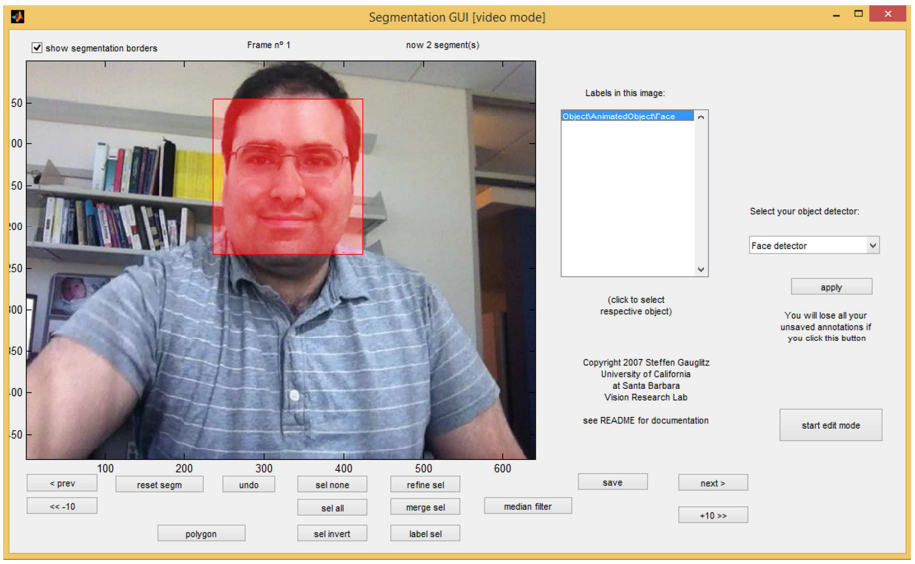

Ilustración 31: Prueba 05 anotar una región

Se entra en "edit mode" y la región que delimita la cara pasa a ser arrastrable. Este paso puede verse en la ilustración 32.

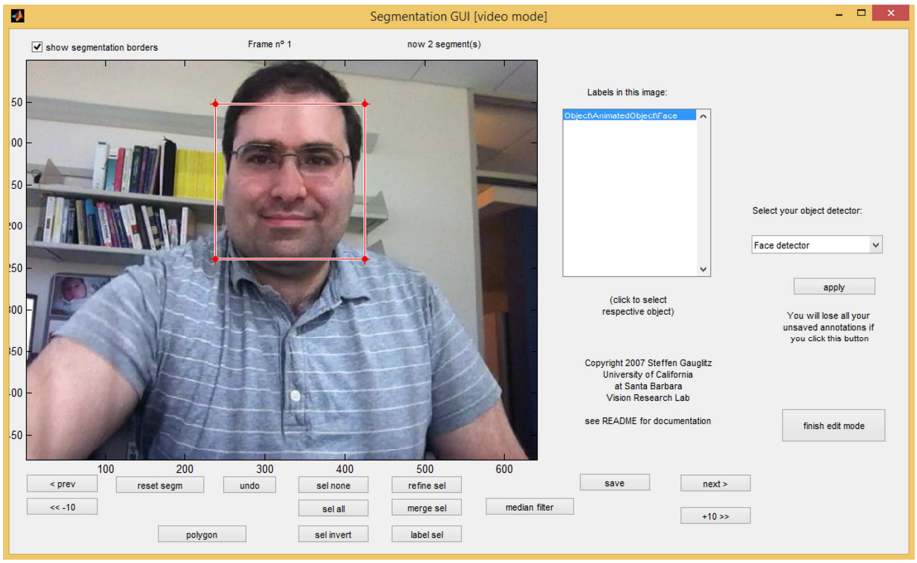

Ilustración 32: Prueba 05 entrar en "edit mode"

Se mueve un poco la región, se pasa a la siguiente imagen y se ajusta la región propagada si es necesario.

En la ilustración 33 se muestra el avance en el fotograma número 10.

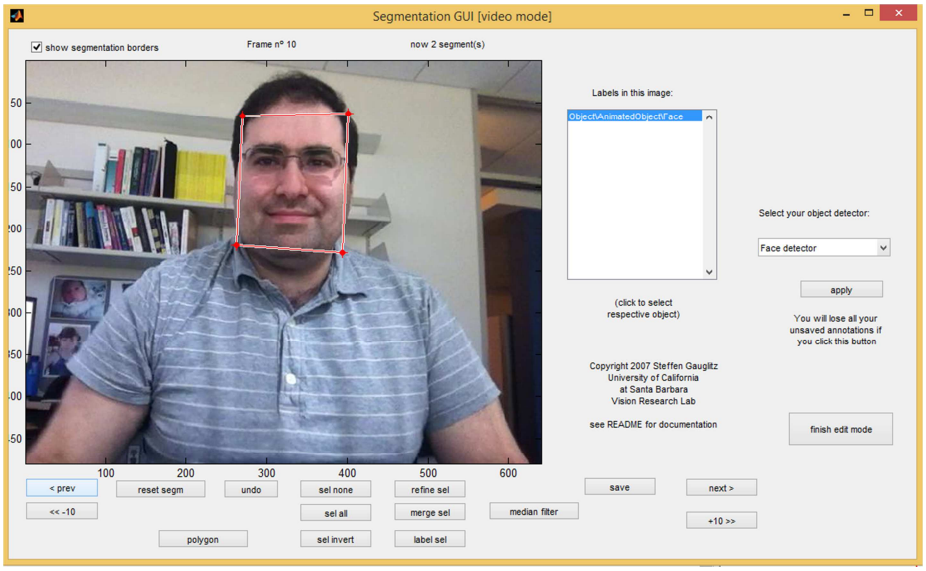

Ilustración 33 Prueba 05 región arrastrada hasta el fotograma número 10

Se sigue sucesivamente hasta llegar al fotograma número 20. La ilustración 34 enseña como se ve la interfaz en este punto.

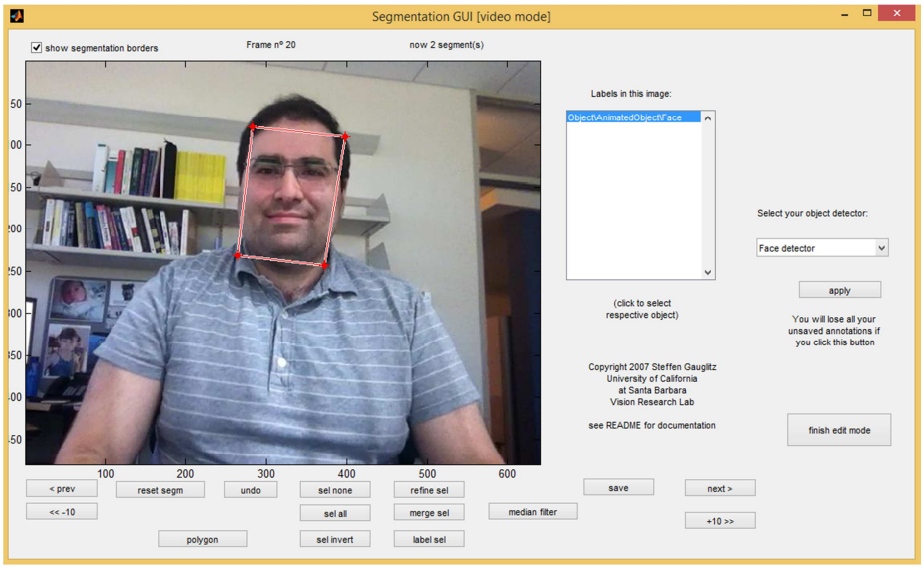

Ilustración 34: Prueba 05 región arrastrada hasta el fotograma número 20

Se sale de “edit mode” y la región propagada aparece en la interfaz de manera adecuada, con su etiqueta asignada y segmentación. La ilustración 35 corrobora que este paso se ha realizado.

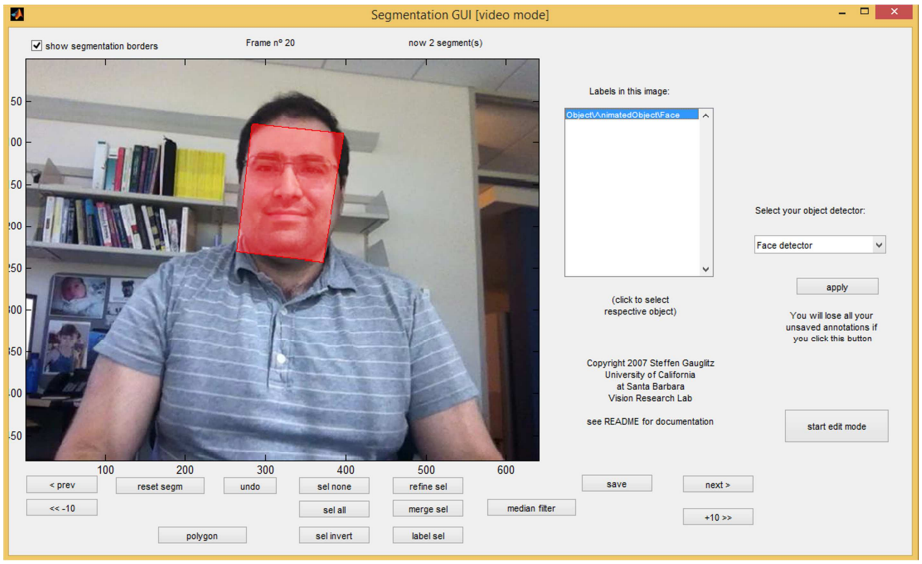

Ilustración 35: Prueba 05 anotación propagada hasta el fotograma número 20

Volver a los fotogramas anteriores hasta llegar al número 1. La región propagada se ha guardado en todos ellos correctamente.

5.1.5.2. Resultado

La propagación de anotaciones es un éxito, todos los fotogramas guardan correctamente la región etiquetada tal y como se ajusta en ellos.

5.2. Comparación entre la herramienta de anotación inicial y la final

La comparación entre estas dos herramientas se hará a nivel de eficiencia a la hora de anotar.

5.2.1. Comparación teórica

Se distinguen dos casos de estudio:

A) Cuando hay que realizar una anotación completa en una imagen en la herramienta de anotación inicial hay que realizar una serie de acciones, que son:
   - Pulsar el botón polygon.
   - Hacer clic en varios puntos de la imagen para delimitar el polígono. Se va a suponer que el polígono va a tener cuatro puntos, ya que así son los polígonos creados por el detector de objetos en la herramienta de anotación final.

- Seleccionar la nueva región.
- Pulsar el botón label sel.
- Elegir una etiqueta de la lista.

A la hora de realizar la misma anotación en la herramienta de anotación final utilizando las anotaciones automáticas se realizan las siguientes acciones:

- Elegir el detector a utilizar de la lista desplegable.
- Pulsar el botón apply.
- Seleccionar una región.
- Pulsar el botón finish.

Mientras que anotar con la herramienta inicial hay que realizar ocho acciones, con la final solamente hay que realizar cuatro.

Además, hay que tener en cuenta que, para anotar dos objetos con la misma etiqueta, la herramienta inicial tendría que realizar siete acciones más que para anotar uno, mientras que la final sólo tendría que realizar una acción más. Esto es porque la herramienta inicial tendría que volver a repetir todo el proceso excepto pulsar el botón label sel, que puede etiquetar varias regiones con la misma etiqueta, mientras que la final solamente tendría que seleccionar una región más en la interfaz de detección.

B) Cuando hay que realizar una anotación completa de un mismo objeto en dos fotogramas seguidos de un vídeo la herramienta de anotación inicial tiene que:

- Hacer el mismo proceso de antes dos veces.
- Pulsar el botón next.
- Elegir la opción de guardar a la hora de cambiar de fotograma.

Lo que hace un total de dieciocho acciones.

Para hacer lo mismo con la herramienta final hay que:

- Realizar las mismas acciones que antes.
- Pulsar el botón start edit mode.
- Mover la región.
- Pulsar el botón next.
- Pulsar el botón finish edit mode.

Lo que hace un total de ocho acciones.

Además, para por cada fotograma más que se anote en estas condiciones, la herramienta inicial debe hacer diez acciones más, que son hacer el proceso de anotación manual, pulsar el botón next y elegir la opción de guardar, mientras que la final sólo debe hacer dos acciones más, que son mover la región y pulsar el botón next.

Aunque esta no sea la mejor forma de comparar los rendimientos de las herramientas, por lo menos deja la idea de que la herramienta de

anotación final es más eficiente a la hora de anotar que la inicial. Los resultados vienen resumidos en la tabla 3.

Caso de estudio	Número de acciones que tiene que hacer la herramienta inicial para realizar el número mínimo de anotaciones	Número de acciones que tiene que hacer la herramienta final para realizar el número mínimo de anotaciones	Número de acciones extra que tiene que hacer la herramienta inicial para realizar una anotación más	Número de acciones extra que tiene que hacer la herramienta final para realizar una anotación más
A	8	4	7	1
B	18	8	10	2

Tabla 3: Resultados de las pruebas teóricas de comparación de herramientas

5.2.2. Comparación práctica

Para la comparación práctica se han realizado una serie de experimentos. En ellos se ha medido el tiempo que tardan la herramienta de anotación inicial y la final en hacer la misma tarea de anotación. Las tareas escogidas para estos experimentos han sido las siguientes:

A) Esta tarea tiene como objetivo comparar ambas herramientas a la hora de anotar varios objetos del mismo tipo en una imagen. Se abre la imagen visionteam.jpg y se anotan las seis caras que hay en dicha imagen. Con la herramienta inicial se hace a través de anotaciones manuales, es decir, se delimitan con polígonos todas las caras y luego se etiquetan todas a la vez con la misma etiqueta. Por otro lado, con la herramienta final se realiza a través de anotaciones automáticas, o sea, se utiliza el detector de caras y se seleccionan todas las caras.

B) Ésta sirve para hacer una comparación entre las herramientas a la hora de anotar varios objetos de diferente tipo en una imagen. Utilizando la misma imagen del caso anterior, se anotan las seis caras, sólo que esta vez lo hacemos como si se trataran de tipos de objetos diferentes, es decir, como si no se los pudiera etiquetar a todos a la vez. Con la herramienta inicial se hace delimitando las regiones con polígonos y luego etiquetando a cada región individualmente. Con la final se utiliza el detector de caras seis veces y en cada una de ellas se selecciona una cara diferente.

C) La siguiente tarea compara los tiempos que tardan las herramientas para realizar anotaciones de un mismo objeto en varios fotogramas de

un vídeo seguidos. Consiste en abrir el vídeo tilted_face.avi y anotar la misma cara en los diez primeros fotogramas. La herramienta inicial tiene que hacerlo manualmente en todos ellos, delimitando la cara con un polígono y etiquetándolo cada vez. La herramienta final debe hacerlo por anotación automática en el primer fotograma, y luego propagar esa anotación hacia el resto de fotogramas usando el mecanismo implementado.

Los resultados pueden verse en la Tabla 4.

Tarea	Tiempo que tarda la herramienta inicial (s)	Tiempo que tarda la herramienta final (s)
A	34	8
B	43	29
C	114	44

Tabla 4: Resultados de las pruebas prácticas de comparación de herramientas

En los resultados puede verse que la herramienta final es más eficiente que la inicial a la hora de anotar en estos casos en los que se pueden utilizar anotaciones automáticas y/o propagación de anotaciones. Además de esto, también se ha realizado una comparación del tiempo que tarda la herramienta de anotación final en recorrer varias imágenes con anotaciones, que hay que cargar y guardar cada vez, y el que tarda en hacer lo mismo pero con varios fotogramas de un mismo vídeo en lugar de imágenes independientes. Se ha realizado la prueba con seis imágenes y se ha tardado unos 13 segundos en hacerlo, mientras que con seis fotogramas de un vídeo se ha tardado 26 segundos. Esto se debe a que las imágenes tienen cada una sus propios ficheros para guardar sus anotaciones, mientras que los fotogramas de un mismo vídeo los comparten, haciendo la tarea de cargar y guardar más duradera.

6. Conclusiones y trabajos futuros

6.1. Conclusiones

Debido a la escasez de herramientas de anotación que permiten anotaciones automáticas y semiautomáticas se ha desarrollado esta herramienta para reducir el esfuerzo humano a la hora de anotar imágenes y vídeos. Al añadir anotaciones automáticas y propagación de anotaciones a la herramienta inicial se ha conseguido que la tarea de anotar sea mucho más eficiente que antes, cuando solamente se podía anotar usando presegmentación automática y anotaciones manuales. La comparación entre la herramienta de anotación inicial y la final lo demuestra.
Además, el hecho de que ahora la herramienta sea capaz de anotar vídeos y no solamente imágenes ha hecho que sea más flexible y proporcione más posibilidades a los usuarios, por lo que interesará a más personas.

6.2. Trabajos futuros

Para trabajos futuros se pueden añadir nuevas características a esta herramienta:

- Permitir la anotación de eventos además de la anotación de objetos. Las acciones que dan lugar a un evento pueden ser anotadas automáticamente y semiautomáticamente mediante detecciones de acciones.
- Añadir anotaciones semiautomáticas que utilicen un algoritmo de template matching y/o de interpolación lineal [1].
- Introducir un mecanismo de aprendizaje incremental que ayude a entrenar detectores de objetos [1].
- Permitir anotar atributos de un objeto [1].
- Poder anotar segmentos de vídeo. A esto podría incluírsele también la posibilidad de reproducir vídeos [2].

Referencias

[1] An interactive tool for manual, semi-automatic and automatic video annotation. Simone Bianco, Gianluigi Ciocca, Paolo Napoletano, Raimondo Schettini. Computer Vision and Image Understanding.

[2] A Survey of Semantic Image and Video Annotation Tools. Stamatia Dasiopoulou, Eirini Giannakidou, Georgios Litos, Polyxeni Malasioti, and Yiannis Kompatsiaris. Multimedia Knowledge Laboratory, Informatics and Telematics Institute, Centre for Research and Technology Hellas.

[3] http://www.w3.org/2005/Incubator/mmsem/wiki/Tools_and_Resources

[4] http://sourceforge.net/

[5] Readme. Proyecto Vision, A6 T5 Reconocimiento Semántico, Herramienta de anotación de sala. Grupo de Tratamiento de Imágenes de la Escuela Politécnica Superior de la Universidad Autónoma de Madrid.

[6] Asignatura Ingeniería del Software. Tercer curso de Grado en Ingeniería Informática de la Escuela Politécnica Superior de la Universidad Autónoma de Madrid.